Chain FL: Decentralized Federated Machine Learning via Blockchain

Caner Korkmaz, Halil Eralp Kocas, Ahmet Uysal, Ahmed Masry, Oznur Ozkasap, Baris Akgun
Koc University, Department of Computer Engineering, Istanbul, Turkey
Emails:{ckorkmaz16, hkocas15, auysal16, amasry17, oozkasap, baakgun}@ku.edu.tr

***Abstract*—Federated learning is a collaborative machine learning mechanism that allows multiple parties to develop a model without sharing the training data. It is a promising mechanism since it empowers collaboration in fields such as medicine and banking where data sharing is not favorable due to legal, technical, ethical, or safety issues without significantly sacrificing accuracy. In centralized federated learning, there is a single central server, and hence it has a single point of failure. Unlike centralized federated learning, decentralized federated learning does not depend on a single central server for the updates. In this paper, we propose a decentralized federated learning approach named Chain FL that makes use of the blockchain to delegate the responsibility of storing the model to the nodes on the network instead of a centralized server. Chain FL produced promising results on the MNIST digit recognition task with a maximum 0.20% accuracy decrease, and on the CIFAR-10 image classification task with a maximum of 2.57% accuracy decrease as compared to non-FL counterparts.**

***Index Terms*—Blockchain, Ethereum, Federated Learning, Machine Learning, Decentralized Federated Learning.**

I. Introduction

Machine learning and its applications are continuously rising in the industry and daily life. Supervised learning is a branch of machine learning, which requires annotated data. In most tasks, collecting and annotating data is a time-consuming and costly process, particularly in the training of deep learning models that require large amounts of data. The data can also include private and sensitive information, and there are ethical concerns about personal data. Therefore, data sharing between different parties is often out of the question, even when there are compelling benefits. This requirement is especially crucial for industries such as medicine and banking since training data often includes personal information, and there are legal restrictions about data sharing [1], [2].

Federated learning allows collaborative model training without openly sharing training data with other participants [2]. It leverages an aggregator, which is an independent centralized server that merges the locally trained models or local model updates. This enables multiple parties to train a model together without sharing any data points, and without allowing original data to be inferred on the aggregator. Several algorithms have been proposed to combine these local models/updates. The simplest one is called Federated Averaging [3], in which each client updates their copy of the current model locally using the gradient computed over their local data. Then, the server averages all these updated model weights to get the updated global model. A very similar algorithm is called Federated Stochastic Gradient Descent (SGD) [3] in which the clients only compute the gradient of the loss (without updating the model). Then, the server aggregates all these gradients and update the global model weights. Since the parties do not want to share their data, the learning must be done partially in the party's local servers. However, this requires the changes in weights and the new global models to be shared across the parties.

The existence of an aggregator in the centralized federated learning systems gives rise to two important challenges. First, the aggregator is a potential single point of failure; if it goes down, the training process fails. Secondly, the parties may not be able to agree on a trusted aggregator [4].

To overcome these issues, several decentralized federated learning architectures have been proposed. Some of these architectures utilize Blockchain technology so that the system can work without a central server to store the machine learning model and update it. This way, even if some miners or some contributing parties' servers fail, the overall system would continue working. However, the blockchain imposes computation and storage limits, which need to be addressed.

Contributions of this paper are summarized as follows. We propose Chain FL, a decentralized federated learning approach which uses a private blockchain network for model storage and aggregating model updates, by utilizing smart contracts. Chain FL system consists of miner nodes and training servers of participating parties. To preserve data privacy, parties get the latest version of the trained model from the blockchain network, train their models locally, and only share the trained weights with the network.

Chain FL implementation uses Ethereum as a private blockchain network, and uses Solidity based Smart Contracts on Ethereum. For fast response times, Proof of Authority is used as a consensus mechanism. Training server implementations are independent of the learning library used, and handle communication with the smart contract on the blockchain network. Moreover, since Ethereum has limits on computation and storage for each smart contract (named gas), Chain FL implements a chunking strategy to allow for large models. This strategy involves dividing the data and the model parameters into multiple small smart contracts and using all of them for the federated learning algorithm.

Chain FL is a decentralized FL architecture which is based on

978-1-7281-8370-1/20/$31.00 ©2020 IEEE

proof-of-authority, is fault resilient and scalable, that allows for asynchronous training, and big data and large models through chunking. Chain FL's novelty is that it is the first FL architecture that combines all of these properties. These are provided by Ethereum smart contracts, and the effectiveness of the approach is demonstrated using deep learning models on MNIST digit recognition task and CIFAR-10 image classification task with a maximum of 0.20% and 2.57% accuracy decrease respectively.

The rest of the paper is organized as follows. In section II, a review of the the related work is provided. In section III, Chain FL system architecture details are described, and in section IV, implementation details are described. The experimental setup and results are discussed in section V and the general conclusion and future directions are presented in section VI.

II. Related Work

Many centralized and decentralized federated learning approaches have been introduced in recent years. One of the centralized approaches by McMahan *et al*. keeps the training data on the contributing parties' devices while training a shared model using aggregated local updates from the devices from which the original data cannot be inferred [3]. In other words, a central server maintains a global copy of the model, and at each time step, the model is sent to the other devices to calculate their updates, which are sent to the server. Then, the server performs the model update by aggregating the local updates. This can be used in many sectors, including finance, healthcare, recommendation systems, not only by large and international companies but also by multiple small and medium-size companies [1]. Moreover, there are federated learning algorithms that are robust to unbalanced and non-independent and identically distributed data coming from different sources [3], and these algorithms can reach similar performances the original machine learning algorithms had at the first place with data from multiple users, devices [3], or institutions [2].

One of these algorithms is called Federated SGD [3]. It works as follows: at each training step, the server selects K clients to use the current global model to calculate the gradients of the loss over their local data, g_k. The server then aggregates all these local gradients and perform the global model update, $w_{t+1} = w_t - \alpha * \sum_{k=1}^{K} \frac{n_k}{n} g_k$ where n_k is the number of data points used by each client, n is the total number of data points, and α is the learning rate. Another commonly used algorithm is called Federated Averaging [3]. Here, the clients do some extra work. Not only they compute the gradients over their local data, but also they update their local copy of the model using their gradients. Then, the server averages the locally-updated models' weights. $w_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$ where $w_{t+1}^k = w_t - \alpha * g_k$

Although these approaches solve the problem of data privacy, they heavily rely on a central server, which aggregates the local updates. Any fault or malicious activity will fail the entire training process. This lead the development of Decentralized Federated Learning Systems in which the training is done on all the connected devices.

Several novel approaches have been proposed to utilize the blockchain technology to perform decentralized Federated Learning. One of these approaches by Kim *et al*. is to store a global model on the Blockchain and its miners and update it based on the local updates provided from the collaborators [5]. Private blockchain ensures that the model updates are shared securely, and the model updates and the model itself are stored in an immutable way. Hence, the complete history of the model is stored in a decentralized way across all the involved devices in the training process. Moreover, if a miner encounters some errors during the process, this will only increase the latency of the system based on the number of malfunctioning minors. Yet, the model data will be maintained on the Blockchain ledger and the overall system can continue the training process successfully. To implement such a system, several approaches utilize previous work on Blockchain. One approach by Ramanan *et al*. builds upon this work that uses a private Ethereum network with smart contracts to store a global copy of the model and aggregate the local updates from the devices to perform a global update of the model [6]. Furthermore, since machine learning models, especially deep learning ones, usually consume much memory storage and require very expensive computational processes, the authors used a model chunking strategy to overcome the technical limitations of the blockchain. However, it still consumes thousands of seconds for each push, as each chunk takes a fixed amount of time to push, and there are thousands of chunks. Consequently, at each training round, the system spends hours on procedures not related to machine learning. Lu *et al*. designed a system architecture for data sharing based on Blockchain while maintaining data privacy [7]. In their system, instead of sharing the requested raw data itself with the requester, the data owners train a data model in a federated fashion on the requested data and share the model with the requester. Then, the data requester can use this model to execute its queries or machine learning predictions with their data, or with newly collected data. Their approach does not store the model or data in blockchain nodes, instead it stores them in a subset of nodes and blockchain contains the metadata about which nodes has them.

As it can be observed in Table I, we can categorize the common decentralized FL approaches by their blockchain architecture, whether they use a round based system and whether they utilize chunking. While custom blockchains allow more flexibility and control, they are not as tested as previously existing blockchain systems. This may result in more security issues in general, and hence adapting a system to use a previously existing blockchain architecture like Ethereum may be more beneficial. Moreover, Proof-of-Work based consensus mechanisms tend to be computationally heavy for the miners as time goes, and custom consensus mechanisms like Proof-of-Quality are not as tested as widely used ones like Proof-of-Authority (PoA). Hence, using PoA allows using a tried and tested consensus mechanism that is not computationally expensive. In addition, federated learning systems can be synchronous or asynchronous based on whether they use communication

Architecture	Centralization	Blockchain	Consensus Mechanism	Smart Contract	Communication Round	Chunking
FedAvg [3]	Centralized	No	N/A	N/A	Synchronous	N/A
BlockFL [5]	Decentralized	Custom	Proof-of-Work	No	Asynchronous	No
BAFFLE [6]	Decentralized	Ethereum	Proof-of-Authority	Yes	Synchronous	Yes
SecureDataSharing [7]	Decentralized	Custom	Proof-of-Quality	No	Asynchronous	N/A
Chain FL	Decentralized	Ethereum	Proof-of-Authority	Yes	Asynchronous	Yes

TABLE I: Comparison of the different Federated Learning approaches in terms of Centralization, Blockchain, Consensus Mechanism, Smart Contract use, Communication rounds, and the use of chunking.

rounds or not. In systems with communication rounds, like BAFFLE [6], parties cannot usually join after a round starts and a minimum number of clients is required. Moreover, round based systems require synchronization between parties (henceforth it is synchronous). In these systems, local updates cannot be aggregated with the global model at arbitrary times, instead aggregation happens at the end of each round. However, they may provide better data privacy as in each round, all parties' new models are averaged before averaging the model with the previous one. We think that for systems designed for better availability for parties, asynchronous version without rounds is more beneficial, and can also be made equivalently data private via differential privacy algorithms. When it comes to chunking, it is unnecessary with custom blockchains since they can be designed to work with large models, but with Ethereum, both the computation and storage is limited in smart contracts. Hence, it is necessary to divide the model into small parts, with separate smart contracts for each of them is required to handle large models (what is large is dependent on the configuration of the private blockchain network).

Similar to these approaches, our Chain FL system is fully decentralized and aggregator-free, powered by the Ethereum blockchain and is asynchronous (without communication rounds). Unlike other systems using Ethereum, Chain FL allows parties to join and contribute updates at arbitrary times. This is achieved by skipping the use of communication rounds. Instead, the communications are performed whenever parties end training, and hence can happen at any given time. Moreover, by not using a round based system and by not having a requirement of minimum number of contributing parties, Chain FL allows parties to continue the training process even if most of the other parties are not available. In addition, Chain FL also utilizes chunking, which allows it to handle large models and big data with our smart contract easily.

III. Chain FL

A. System Architecture

Overall architecture consists of at least one miner for each contributing party hosted on their own servers, with separate machine learning servers for doing the training and uploading the model to the miners. Miners run the smart contract code previously deployed on the network to apply the federated learning update step. Since the contract runs on all of the

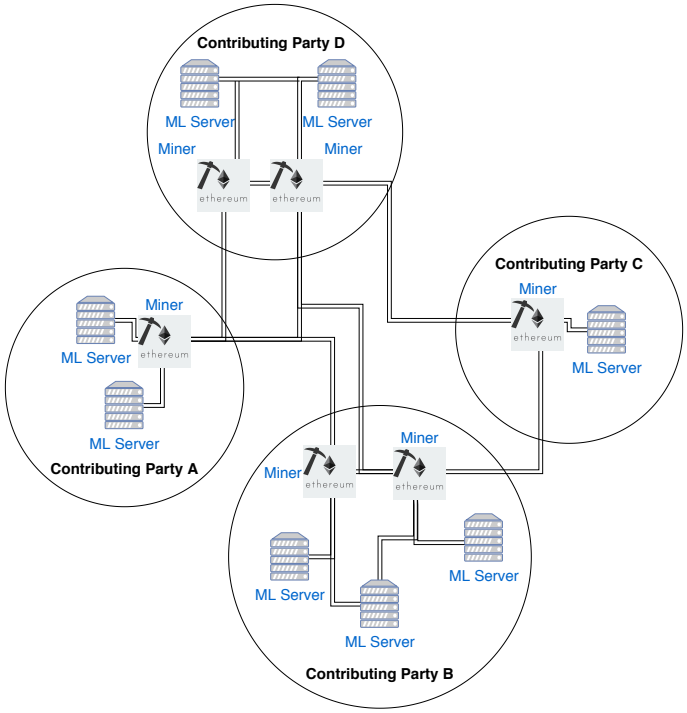

Fig. 1: Illustration of the sample system architecture with four contributing parties.

miners, our system to works even with failures of one or multiple miners owned by different parties. In addition, by utilizing properties of the blockchain networks, Chain FL provides an immutable history of the global model for each party, stored in every miner in the system. Any party can access the most recent model, as well as the history, from the smart contract deployed in the private blockchain network. A sample system architecture diagram with four contributing parties is depicted in Figure 1.

B. Private Blockchain Network

Blockchain network consists of miner nodes which are run by participating parties. Ethereum by itself does not allow storing custom data in blockchain blocks. For this reason, we used Smart Contracts on Ethereum, which allow storing data in the blockchain blocks, and executing code in miners that can alter the stored data. Smart contracts are compiled code blocks

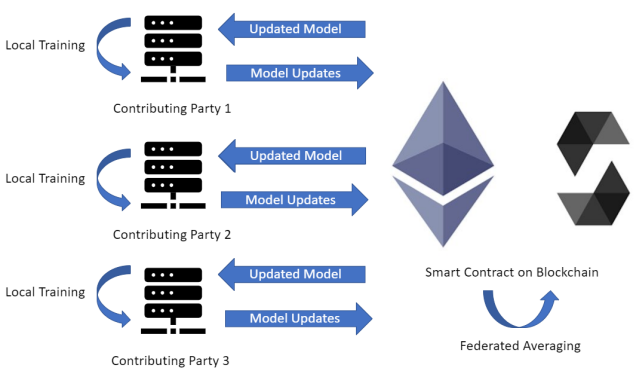

Fig. 2: Training Process in Federated Learning with Blockchain.

stored on blockchain blocks, and when a miner gets a request for execution of a smart contract method, all miners retrieve the code from the blockchain, execute it and try to store the result. Since everyone executes the code, they need to agree about which value to store (Smart contracts are deterministic but there can be malicious or faulty parties), and this is achieved through consensus mechanisms. Traditionally Proof of Work is a highly used blockchain consensus algorithm, however, it takes longer and longer as the system grows, which is not wanted for our scenario. Hence, we used Proof of Authority, where nodes put their identity as a stake. There are validator nodes which can decide on the accepted value, and every node can be a validator by calculating correctly and getting a good reputation. Moreover, validator nodes can lose their validator status by getting a bad reputation from incorrect calculations. Overall, this allows the complete system to reach a consensus about which value to accept, and this takes a fixed predetermined amount of time set by us. Thus, we used a private Ethereum network with smart contracts, and Proof of Authority consensus mechanism.

C. Federated Learning with Blockchain

Federated Learning allows contributors to privately hold training data in their own devices or servers. To allow these parties to train a model together, the model updates (model weights) from local training steps are shared instead of the actual data. The complete process for a single training round can be seen in Figure 2. In the beginning of training, the most recent model is taken from the private blockchain network to the corresponding contributing party servers. These servers locally train the model using new or previous data. After the training step, the model updates are sent back to the blockchain network by executing the corresponding method in the smart contract. As a final step, the local model updates from all contributors are merged into the stored weights in the smart contract utilizing a customized version of federated averaging algorithm.

Federated Averaging is a process of calculating weighted average of model weights. In order to assign a weight to updates from each participant, we use the number of data points used for this local update step. Thus, all updates taken from participating parties are weighted by using the number of data points used in training. In normal federated averaging, in a single round, updates from contributing parties are averaged with weights and this is taken as the new model. However, this requires the update step to take place at the same time for each contributor, or storing the update until the round finished. This was not feasible for the Smart Contract based version, and moreover, this is more suited for a single party multi device federated learning, and is less suited for multi-party federated learning. Instead, we used Online Federated Averaging, where we start with zero weights and use Equation 1 to update the model. Unlike the classic version, this eliminates the need to wait for each contributor to send their own part by updating the weights in an online manner when there is an update. Moreover, for a single training round, this is mathematically equivalent to the normal federated averaging. The equation also takes new data points into consideration for calculating the average and total number of data points correctly. Here, new data points are the ones being used for the first time. This is mainly useful to support increasing number of data points from parties.

$$w_{i+1} = \frac{(n_i - o_i^*)w_i + (n_i^* + o_i^*)w_i^*}{n_i + n_i^*} \qquad (1)$$
$$n_{i+1} = n_i + n_i^*$$

Equation 1: Online Federated Averaging update step where w_i denotes current weights of trained model after ith averaging step, n_i denotes the total number of unique data points used to train the current model at ith step, w_i^* denotes weights of the locally trained model, o_i^* denotes the number of data points that was already added to n_i and were used to obtain the weights w_i^*, and n_i^* denotes the number of novel data points that was added to the local dataset after the ith step that were also used to obtain the weights w_i^*.

With this approach, models are fetched from the smart contract before local training. Each party can pull the model from any step, train it locally and send it for aggregation. Even though the best case scenario is every party always using the latest model for training, sometimes multiple parties can pull the model at the same time, train it locally, and then send it to the smart contract at similar times. In this scenario, models that aggregated originate from the same point, but they are aggregated into a newer global model. This doesn't affect the training process since it is similar to the round based approach, where models are trained using the same model from the previous round, and then all are aggregated into that model. Hence, this system allows parties to train at the same time without waiting for other parties' training to finish as expected (and like in other systems), but also allows the parties to train sequentially at different times.

Training Step	Small NN Classical	Small NN Federated	Large NN Classical	Large NN Federated
20 Epochs 1/3	-	95.01	-	98.38
20 Epochs 2/3	-	95.91	-	99.00
20 Epochs	96.58	96.40	99.42	99.16
40 Epochs 1/3	-	96.43	-	99.20
40 Epochs 2/3	-	96.65	-	99.26
40 Epochs	96.80	96.75	99.48	99.28

TABLE II: Accuracy comparison of equivalently trained small and large neural network models with classical and federated training on MNIST dataset.

IV. Chain FL Implementation Details

A. Smart Contract

We developed our smart contracts using Solidity, which is the programming language used for developing smart contract on Ethereum networks. Since Ethereum Virtual Machine (EVM) does not have floating point operations, we use 128 bit integers consisting of 64 bit numerator and 64 bit denominator part. These are converted to and from floats in the machine learning client part of the system. Moreover, due to lack of multidimensional arrays with dynamic sizes in EVM, we store the weights as a single flattened array, also converted to and from on the machine learning client. Federated learning update step is applied on the flattened array of 128 bit integers using the Eq. 1. Moreover, it marks an event in the blockchain history containing the updated model, which allows one to get previous models from the blockchain network. In addition, we seperated our models into multiple small contracts to make sure both computation and data storage stays under the gas limits. For finding how much data should be in a chunk, one can set the private blockchain gas limit to different numbers, and try different size contracts. Based on this, an approximately maximum gas limit and a corresponding number of elements in a chunk can be found. For example, even though this would work with a smaller gas limit, we were able to utilize 50000 parameters in a single smart contract with a gas limit set to 0xE575CB441C. In addition, since chunking will result in many small smart contracts, it is better to set the time to mine next block in PoA consensus mechanism to a small number, less than 5 seconds. Otherwise, uploading new local models take long amounts of time.

B. Training Client

The training client consists of two main parts: the communication part, and the training part. While communication part handles uploading the model to the smart contract and receiving previously uploaded models, the training part handles doing model training using libraries like pytorch or tensorflow. Communication part of the local client handles the transmission of the new model and data point numbers to the miners. This is achieved by using the ‘web3py‘ library for python, which allows us to call methods on smart contracts that were deployed on the blockchain network. This call is similar to a transaction, and each miner runs the averaging function in the smart contract to find the new weights. Moreover, since we use int128 based real numbers in the contract, this part also handles conversion from float to int128 and vice versa. In addition, this part divides the flattened data into multiple arrays for compatibility with chunking.

The training part handles loading the data, testing, flattening and unflattening the model, receiving the model from communication part and sending the new model to the communication part. It tries to receive the most current model from the blockchain, runs for the given amount of epochs, tests after each epoch, then sends the new model, gets the average one, and tests also after averaging.

V. Experimental Setup and Results

A. MNIST Experiment

The experiments on Modified National Institute of Standards and Technology (MNIST) dataset consist of two different experiments, one with a small model and one with a larger model. First one is a fully connected network with 768 input nodes, 32 hidden nodes with Rectified Linear Unit (RELU) activation, 0.2 Dropout, 10 output nodes with LogSoftmax Activation, trained with Negative Log Likelihood Loss, Adam Optimizer (0.001 initial learning rate, 0.9, 0.999 as beta values) and Step based Learning Rate Scheduling (Step 5, gamma 0.5) and mini batch with size 256. The second one is also a fully connected network, with 784 input nodes, 512 hidden nodes with ReLU activation, 0.2 Dropout, 128 hidden nodes with ReLU activation, 0.2 Dropout, 10 output nodes with LogSoftmax Activation, trained with the same loss, optimizer and parameters. The first one has 25,450 parameters, and the second one has 468,874 parameters in total. We used the original MNIST train-test dataset split (60’000 for training, 10’000 for testing). Classical (non-federated) training were performed on both models with 20 and 40 epochs on the MNIST training split. Then, the data was divided into 3 parts, and a 20 epoch training were performed with 1st data part, averaged, 20 epochs with 2nd part, averaged, 20 epochs with 3rd part, averaged (resulting in equivalent training to classical with 20 epochs), and then this process was repeated once more (resulting in equivalent training to classical with 40 epochs). The model was evaluated after each average step using the accuracy on complete MNIST test split.

Table II presents the results of this experiment (in the table, ”-” means not applicable, as classical version is always trained with the complete training dataset). There is a slight difference between the performances of federated training and classical training in both neural networks. As the number of epochs increase, the performances of federated learning converges towards to the performance of the non-FL counterpart. The reason for slower convergence is due to the fact that data is split between multiple parties. Relations between data in different splits will only be learnt when there is a averaging step, hence more communication is required to converge to the

non-FL counterpart scores. However, even after 20 epochs, the results get very close to the classical, showing that Chain FL is effective for fully connected neural networks.

B. CIFAR-10 Experiment

The second experiment we run was on the Canadian Institute For Advanced Research (CIFAR-10) dataset, which consists of 32x32x3 sized 60'000 images (50'000 for training, 10'000 for testing), and 10 output classes [8]. The model we used for this experiment is similar to a VGGNet architecture without the full dense hidden layers at the end, together with Batch Normalization, L2 regularization and Dropout [9]. It consists of convolutional layers with 3x3 kernels, 1 padding. There are 3 of such layers with 32 outputs, ReLU activation and batch normalization, then a 2x2 max pooling layer, then dropout of 0.2, then 2 of such layers with 64 outputs, ReLU activation and batch normalization, then a 2x2 max pooling layer, then dropout of 0.3, then 2 of such layers with 128 outputs, ReLU activation and batch normalization, then a 2x2 max pooling layer, then dropout of 0.4. Then, they are flattened, and connected to the 10 output nodes, together with a log softmax afterwards. This network has 317,706 parameters in total. This is trained with Negative Log Likelihood loss, and optimized with AdamW variant of the Adam optimizer, with learning rate 0.001, betas 0.9 and 0.999, and weight decay of 0.01 for L2 regularization [10], [11].

Total training is done for 200 epochs, and max learning rate is reduced by a factor of 0.1 at epochs 100 and 150 in both scenarios. Moreover, data augmentation consisting of random crops with padding 4, random flips and normalization is used on the training data at each epoch, and the same normalization learnt from training data is used on the test data as well. We used a batch size of 128, and divided the training data into 5 random parts of size 10'000 for the federated case. These indices were divided randomly, without explicitly balancing the output label distribution. In federated scenario, first 50 epochs is done iteratively for each party, then another 50 epochs is done, and afterwards, 50 epochs with learning rate multiplied by 0.1 and another 50 epochs with learning rate again multiplied by another 0.1 is performed. We followed this process to provide a similar test scenario to the classical training.

The Table III shows the test accuracy of both classical and federated learning scenarios with respect to epochs and our random splits. The federated training performs worse than the classical training as expected. Yet, the performance of federated training gets closer to the performance of classical training after the averaging step. The decrease in accuracy is larger in the CIFAR-10 experiment as compared to MNIST experiment (i.e convergence is slower. One main reason is that we utilized 5 splits instead of 3, slowing the convergence speed and requiring more communication to learn the same relations between data points. Moreover, another reason is that CIFAR-10 is a more complex task than MNIST. Despite these points, the results show that Chain FL is still viable for CNN based architectures on more complex tasks, with a accuracy score decrease of 2.57% in CIFAR-10 dataset.

Training Step	Classical	Federated
50 Epochs	89.22	86.29
100 Epochs	90.26	87.95
150 Epochs	91.57	88.82
200 Epochs	91.65	89.08

TABLE III: Accuracy comparison of equivalently trained neural network models with classical and federated training on CIFAR dataset

VI. Conclusions and Future Work

Common federated learning solutions use a centralized approach for storing and aggregating the models, which creates a single point of failure. We provide a blockchain based decentralized solution allowing multiple parties to do federated learning without using any central servers. This doesn't effect results when compared to traditional federated learning as it uses the same update steps. However, results gets slightly worse when compared to classical machine/deep learning, as expected. We have seen a maximum of 0.20% accuracy decrease with MNIST dataset divided into 3 different parties, and a maximum 2.57% accuracy decrease with CIFAR-10 dataset divided into 5 different parties. The results are promising and show that decentralization and blockchain technology can be applied to federated learning.

The solution can be improved in many different ways, and two of the most important ones are: fault tolerant federated learning for preventing poor quality data or malicious parties impairing the model, and better handling of large models with other decentralized storage options.

A. Fault/Data Tolerance Improvements

Data quality of individual parties affects the collaborative training process. Moreover, faults occurring at the local code or in between the smart contract and local code may result in worse updates than usual. In addition, there may be Byzantine fault tolerance issues, like malicious parties trying to make the model worse. Automatic detection and prevention of these scenarios play an important role for a multi-party federated learning system. Poor quality updates resulting from poor data quality, faulty updates and malicious updates can all be prevented either by using a centralized test dataset to check update performance [12], [13], or by using decentralized test datasets other parties have [14]. The first one has a simpler implementation, but it is not scalable and requires parties to share some parts of the data as the test dataset. The second one is harder, and requires sending updates to parties and getting test scores back before updating the model, but is more scalable and requires no data sharing.

B. Decentralized Storage Alternatives

Even though chunking is a fine strategy for storing large amounts of data in a private Ethereum network, it results in working with many smart contracts, and also makes the time for receiving and uploading models much longer than required. The

chunking requirement is particularly due to the model being stored in each and every miner on the network. If this amount of redundancy is required, this is a good option. However, in practice, systems with less redundancy can also be used, and they would be faster in runtime and also storage costs.

There are different alternative ways to handle decentralized storage, and we think that larger models can also be stored in a decentralized way more effectively with systems designed to be able to handle large amounts of storage. A custom blockchain implementation is one of the promising solutions to this [3], [7]. Moreover, it is possible to store the model redundantly by each participant, without storing it fully on every single node, reducing overall storage requirements while providing similar guarantees [7]. This requires previous knowledge about the parties, which is not related to or feasible for usual blockchain networks, but it is feasible for a custom created one. Moreover, other consensus algorithms like Hashgraphs could be implemented and tested [15]. Other fully decentralized algorithms could also be implemented, like Gossip Aggregation algorithms for sharing the model between parties in the training phase [16], [17]. Algorithms with centralized aggregators that are internally implemented with fault tolerant distributed systems can also be alternatives where decentralization is not required.

C. Additional Experimental Studies

To be able to use such a system in a production scenario, more experimental studies should be conducted beforehand. Most important ones include detailed experiments comparing with other centralized and decentralized federated learning systems with respect to decrease in accuracy, fault and data tolerance, response times and communication overheads. Moreover, different machine learning and deep learning setups could be tested to observe how the performance of different architectures gets affected by ChainFL. Realistic simulations in churn scenarios where parties join at different times and may disconnect due to network and hardware failures can help better understand performance of ChainFL.

REFERENCES

[1] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, Jan. 2019. [Online]. Available: https://doi.org/10.1145/3298981

[2] M. J. Sheller, G. A. Reina, B. Edwards, J. Martin, and S. Bakas, "Multi-institutional deep learning modeling without sharing patient data: A feasibility study on brain tumor segmentation," pp. 92–104, 2019.

[3] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," vol. 54, pp. 1273–1282, 20–22 Apr 2017. [Online]. Available: http://proceedings.mlr.press/v54/mcmahan17a.html

[4] A. G. Roy, S. Siddiqui, S. Pölsterl, N. Navab, and C. Wachinger, "Braintorrent: A peer-to-peer environment for decentralized federated learning," *CoRR*, vol. abs/1905.06731, 2019. [Online]. Available: http://arxiv.org/abs/1905.06731

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

[6] P. Ramanan and K. Nakayama, "Baffle : Blockchain based aggregator free federated learning," 2019. [Online]. Available: http://arxiv.org/abs/1909.07452

[7] Y. Lu, X. Huang, Y. Dai, S. Maharjan, and Y. Zhang, "Blockchain and federated learning for privacy-preserved data sharing in industrial iot," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 6, pp. 4177–4186, 2020.

[8] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[9] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, cite arxiv:1409.1556. [Online]. Available: http://arxiv.org/abs/1409.1556

[10] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[11] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *7th International Conference on Learning Representations ICLR, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=Bkg6RiCqY7

[12] L. Muñoz-González, K. T. Co, and E. C. Lupu, "Byzantine-robust federated machine learning through adaptive model averaging," 2019. [Online]. Available: http://arxiv.org/abs/1909.05125

[13] J. So, B. Guler, and A. S. Avestimehr, "Byzantine-resilient secure federated learning," 2020. [Online]. Available: http://arxiv.org/abs/2007.11115

[14] S. Guo, T. Zhang, X. Xie, L. Ma, T. Xiang, and Y. Liu, "Towards byzantine-resilient learning in decentralized systems," 2020. [Online]. Available: http://arxiv.org/abs/2002.08569

[15] L. Baird and A. Luykx, "The hashgraph protocol: Efficient asynchronous bft for high-throughput distributed ledgers," 2020.

[16] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *44th Annual IEEE Symposium on Foundations of Computer Science Proceedings*, 2003, pp. 482–491.

[17] I. Hegedűs, G. Danner, M. Jelasity, and L. Ricci, "Gossip learning as a decentralized alternative to federated learning," in *Distributed Applications and Interoperable Systems*, J. Pereira, Ed. Cham: Springer International Publishing, 2019, pp. 74–90.