Using Blockchain for Decentralized Artificial Intelligence with Data Privacy

Anand Surendra Masurkar, Xiaoyan Sun, and Jun Dai
Department of Computer Science
California State University, Sacramento, CA, 95819, USA
Email: anand.masurkar188@gmail.com, xiaoyan.sun@csus.edu, jun.dai@csus.edu

2023 International Conference on Computing, Networking and Communications (ICNC) | 978-1-6654-5719-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICNC57223.2023.10074247

***Abstract*—Artificial Intelligence (AI) solutions require model training, which traditionally needs local datasets to be uploaded to a centralized server. However, centralized learning usually creates issues in terms of data privacy, data control, and data security. To address these issues, decentralized AI, especially federated learning, is proposed to achieve no data sharing or data exchange across servers and decentralized devices. This paper proposes a new decentralized AI paradigm based on blockchain, which goes beyond the decentralized federated learning to require no model pooling or orchestrating on any central server. That is, we propose further decentralization by dismissing the central orchestrator in federated learning through the usage of blockchain (e.g., Ethereum in this paper), which is inherently a decentralized, distributed, peer-to-peer and autonomous mechanism. We implemented a prototype to demonstrate the idea, in which models are trained on data contributors' ends. Instead of moving models to a central place for combining (such as in federated learning), models are trained independently on data contributors' sides, and flow to the decentralized blockchain for updating. In this process, smart contracts on Ethereum provide specifications required for model creation. In addition, data contributors are rewarded Ether coins for their contributions to train the model with their data. This also incentivizes data contributors to provide data for AI training while protecting their data privacy.**

***Index Terms*—Artificial intelligence, Blockchain, Data privacy**

I. Introduction

Today artificial intelligence (AI) [1] has shown great potentials to solve complex problems, such as pattern recognition, natural language processing, image processing and video processing. AI solutions usually require data collection across multiple edge devices or sites for model training in machine learning or deep learning [2]. Traditional methods in AI require the collected data to be uploaded to a centralized server for processing. This type of AI solutions are categorized as centralized learning, and they are known to suffer from issues associated with data centralization.

1) *Data privacy*: processing collected data on a centralized server means data exposure to data collectors other than data owners. This kind of exposure, inherently unintentional disclosure of personal information, is unexpected for sensitive data such as healthcare information, and it imposes a huge threat on the ownership and privacy of such data.

2) *Data control*: once data is transmitted to the central server, data contributors lose control over their data. That is, they cannot further control the usage of their data. The data can be shared with third parties for collaborative efforts, or be traded with other groups for benefits. When access rights are inappropriately configured, the data can also be revealed to unauthorized entities.

3) *Data security*: the data access issue on the central server goes even wilder when malicious actions (such as hacking) are involved. Specifically, a data breach occurs when a target server is compromised, and hackers can steal its data without being noticed. For example, in the 2013 Target data breach, 40 million credit and debit records and 70 million customer records are stolen by hackers. Similarly, the traditional AI scenario centralizing vast data items on a server makes a perfect victim of data breach attacks, raising serious concerns that the data used for centralized modeling could be leaked to malicious entities as well.

To address the above issues, *decentralization* becomes a key to preserve privacy in AI paradigms. A prominent example is the decentralized federated learning, in which the points of model training based on local data are separated from the central point of model combining for a global model. To achieve this, the technique targets to aggregate models instead of aggregating data (that can be unevenly distributed and heterogeneous). Specifically, the decentralized AI scenario allows a model distributed from the central server to individual edge devices (a.k.a. clients) or in advanced versions a model uniquely defined by the individual client. Each client trains the model based on its own local data, and then uploads the modeling results to the central server for re-combining. In this paradigm, it is the models that flow between clients and the central server, not data. Thus, no data sharing or exchange across decentralized edge devices are needed for the learning, and the above issues can be effectively avoided.

However, we find that thus far **the decentralization is not fully achieved** in the above decentralized AI paradigms. A central point is still needed for model re-combining, which could be leveraged by hackers to infer the model parameters or even client's local data. The possibility and feasibility were pointed out in [3] and investigated in [4]. This paper is an effort to solve this problem by replacing the central model orchestration point with the blockchain mechanism, which was born to be decentralized, distributed, peer-to-peer and autonomous. As a proof of concept, we specifically choose Ethereum as the blockchain platform, based on which smart contracts can provide specifications required for model cre-

ation, and implement a prototype to demonstrate that models can be trained on data contributors' ends sequentially and flow to the next one through Ethereum for updating. Benefiting from this blockchain-based design, the central point for model pooling and orchestration is fully dismissed, and at the same time data contributors can be rewarded Ether coins for their contributions to train the model with their data. Data consumers who need data to train their AI models will need to provide the monetary incentives to data contributors. Our approach therefore proposes the blockchain-based AI paradigm to give data contributors both data privacy guarantee (through full decentralization) and monetary incentives to contribute data for AI training. In addition, we also limit the cost for participants by using IPFS to store the full models, instead of pushing everything on the blockchain.

II. Related Work

A. Blockchain Technologies

Blockchain is a distributed ledger. It stores data permanently. It is immutable. Blockchain is nothing but a chain of connected blocks where each block contains a hash of the previous block. As each block contains a hash of the last block, any change in any of the blocks alters all the subsequent blocks following it. Any alterations will result in affecting the subsequent blocks with the new hash.

There are different types of nodes in the blockchain. A node with a full downloaded live up-to-date copy of the blockchain is a *full node*. It uses a consensus algorithm to determine the order of transactions on branches in geographically separated full nodes. Full nodes validate the history of blockchain. A node that only store a portion of the blockchain is a *light node* or thin node. It holds the block header of the previous transaction and confirms the validity of the blockchain. Block header contains information about the previous block to which it is hashed, nonce, and the time it was hashed.

There are three primary types of blockchains: *public*, *private*, and *hybrid* blockchain. Public blockchains are open, and any party can participate as developers, users, governments, institutions, banks, or miners. They are entirely decentralized, and no individual or third party controls the transactions in the blockchain. All public blockchains have a token associated with them to incentivize participants in the network. The second type of blockchain is private blockchain, also known as permissioned blockchain. These are somewhat centralized, and the owner or group of owners of the blockchain has control over the individual participants. Transactions in this chain are private and accessible only to the participants who are permissible to be a part of the network by the owner of the chain. Lastly, we have hybrid blockchain, which combines features of private and public chains to include privacy benefits and security and transparency, respectively. This type of chain allows choosing what data the company wants to keep private and what can be publicly available. Research communities have started exploring Blockchain based federated learning [5]–[7]. In this paper, we use Ethereum as the blockchain-based decentralized network. On top of blockchain features, Ethereum has smart contract functionality. This contract can be viewed as rules or statements which should hold true or lies within the defined set of parameters in the smart contract.

B. Preserving Privacy in AI

Recent researchers have made efforts to safeguard user data privacy in developing artificial intelligence models. In 2016, Google published a blog on a privacy-preserving distributed learning framework called secure federated learning [8]. It gives details on how conceptually this framework would work without centralizing the training data. On individual devices, machine learning is performed to get a locally trained model. These trained models from each user are said to be immediately averaged at the central system, and no individual update is stored in the cloud. Google has also incorporated this technique in their products like Gboard [8] to provide on-device inference and learning capabilities. According to the blog, the company is still testing federated learning on the android google keyboard. In this paper, it is proved to be very useful in developing the foundation for the current system.

The research done by Yang et al. explains challenges faced in AI and further proposes a framework for federated machine learning [9]. It is based on three distribution characteristics of the data which are required for developing learning models. It describes horizontal federated learning, vertical federated learning, and federated transfer learning. In this paper, it helped to understand how isolated data can be used for model training. The authors proposed privacy preserving federated learning solutions for traditional machine learning logistic regression models. Our system trains a neural network model locally on distributed nodes and can also be used to develop traditional machine learning models with few modifications.

Ghodsi et al. explains the SafetyNets protocol developed in order to verify the execution of neural networks on an untrusted cloud storage [10]. It also provides an example where a server changes the classification output intentionally and how SafetyNets protocol prevents incorrect computations from such a malicious server. However, this protocol is tailored for a specific class of deep neural networks, i.e., those that can be represented as arithmetic circuits. However, in our approach, we use decentralized nodes (clients) for model training; therefore, we do not require any cloud storage, thus eliminating the situation to trust any unreliable cloud storage.

Trask et al. developed a generic privacy-preserving deep learning framework and used a concept called virtual workers in PyTorch [11]. These virtual workers communicate with each other in a secure way, using techniques called differential privacy. The results of this approach are tested on Boston Housing and Pima Diabetes datasets and have performance overhead. Our paper aims to perform model training by participating clients via collaborating in a sequential manner. As there is no need for a centralized aggregator for the final model, we have not developed any differential privacy techniques but can be developed on top of the current system in the future and see how the system behaves with encrypted training on decentralized nodes.

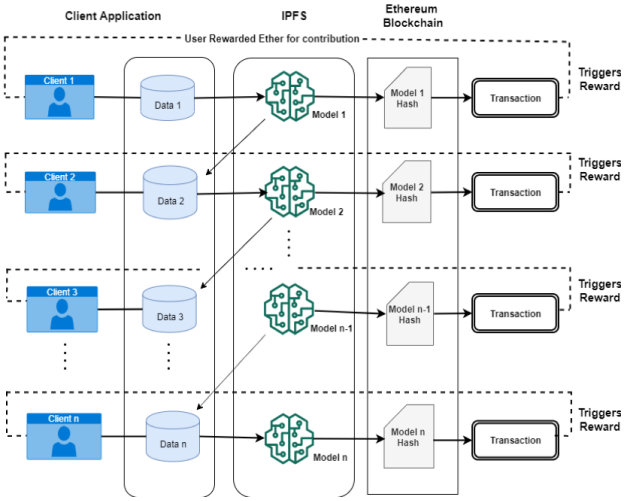

Fig. 1. Overview of Approach

Wood et al. describes Ethereum, a secure decentralized transaction ledger with smart contract functionality [12] [13]. Smart contract can be used to provide an incentive mechanism for the clients to contribute. [12] provides in-depth details on the design, issues, and opportunities of the Ethereum.

III. Overview of the Approach

Fig. 1 shows the overview of our proposed approach. It consists of three major components: client application, IPFS (i.e. the interplanetary file system), and Ethereum blockchain. The clients, as data contributors, can contribute their data through client-side application. The IPFS is used to store the trained models. The Ethereum blockchain can record transactions and issue rewards to clients/data contributors.

The process initiates from the client-side, where each individual client holds its own private data. This private data is accessible only by the data owner. When the first client starts the training, the system will begin the initial configuration and compilation of the model. For example, in Fig. 1, when *Client 1* initiates the process, the system will configure and start to train the AI model. Upon completion of the process, the trained AI model is then transferred to a distributed file system. This file system is known as the interplanetary file system (IPFS). When the IPFS stores a file, it returns the cryptographic hash of the file. The hash can be used to access the file content from IPFS at any later stage. This hash of the model is recorded in the Ethereum blockchain as a transaction. In our work, the AI model trained by *Client 1* is stored in IPFS. Once a transaction is complete, it triggers a reward for *Client 1* with some ether amount. Further, *Client 2* will train on its data. Our system then checks if the hash of any previously trained model is available in the Ethereum blockchain. If yes, *Client 2* will obtain the previously-stored model from IPFS and train the model on its own data. Upon completion of model training, *Client 2* will also store the model to IPFS. Again, the hash value of the model is recorded in the blockchain, and *Client 2* is rewarded for its contribution. This process continues until *Client n* completes its training. The threshold on the number of training participants to train a complete model can be specified in advance. The final model is stored in IPFS.

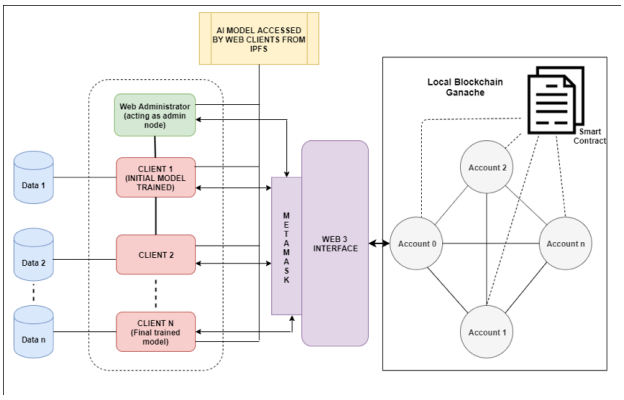

Fig. 2. System Design

IV. System Design

This section elaborates the design of a system as proof of concept. As shown in Fig. 2, the whole system can be divided into four major components: *application*, *IPFS*, *Ethereum blockchain*, and the *Metamask and web3 interface*. In details, the client-side application is where the administrator and data contributors specify parameters and contribute data; IPFS is an interplanetary filesystem to store files in a distributed network; the Ethereum blockchain network (Ganache local Ethereum blockchain for development purpose) contains smart contract functionalities that implements business logic for the application; and the Metamask and web3 interface enables clients to interact with the ganache local blockchain.

Actors in The System. There are two types of actors in the system: *Administrator* and *Clients* or *Data Contributors*. Both actors are distributed nodes on the Internet and have distinct roles in the system. Actors need the Ethereum account to connect to the decentralized network and use features provided by the system. Clients have decentralized data distributed across various clients. This data is private and specific to their data usage. In order to train the AI model participating clients, we need an Ethereum account. This account is imported in the Metamask client browser extension to connect to the Ganache Ethereum Blockchain. This account identifies the client, and all activities performed by the connected account can be tracked in smart contracts to provide necessary access to system functionalities. Clients can train the AI model locally and have *save* and *retrieve* functionalities to and from the IPFS with the help of smart contracts. The second actor in the system is known as the administrator. The administrator has different responsibilities, such as setting model specifications in the smart contract. This allows clients to access these specifications to configure and build the AI neural network for training on their ends without centralizing the data. The administrator does not hold any private data and cannot participate in the training process.

MetaMask. MetaMask is a wallet that secures a private key. It is an account management tool that enables interaction with the Ethereum blockchain network. It can also connect

to the local blockchain network (the testnet) for development and testing purposes before actual deployment to the mainet. MetaMask is deployed as a browser extension and triggers a popup when any client connected to the Ethereum updates the state of the smart contract. The update can also be viewed as creating a transaction. It signs the transaction with the private key of the client or the administrator, depending on which account is active in the MetaMask.

Web3 interface. Web3.js is an Ethereum JavaScript API. It is basically a collection of libraries which helps external accounts or clients to interact with local or remote Ethereum node. It uses HTTP of IPC connection.

IPFS. It is a distributed system for storing and accessing files, websites, and data. Unlike traditional servers that serve files from a dedicated server, IPFS serves files in a distributed way. It finds the requested file based on the file content, rather than file location. Content identifier can point to many different types of data [14] [15]. In our system, IPFS stores the binary data file representing the weights of the AI model.

Ganache Blockchain. Ganache is a tool that can be used to create a private and local blockchain on a local machine. It enables developers to deploy and test their developed smart contracts before deploying to the mainet or testnet. Ganache provides ten external accounts, which in our system can be used for clients and administrator for training purposes.

Smart Contract. The blockchain can implement code execution using a smart contract. A smart contract is where all the business logic of the application is deployed. The code in a smart contract will be executed when predetermined conditions are met. The system uses a smart contract to initialize the neural network model using specifications and to validate whether a client has already contributed to the training on its privately-owned data.

V. Implementation

A. System Pre-requisites

For the implementation, we used Node Package Manager (NPM) and Truffle Framework. Truffle framework allows us to build decentralized applications on the Ethereum blockchain. It provides tools to write smart contracts in the Solidity programming language. We can also run tests on smart contracts before deploying them on the blockchain. It also includes a place to develop a client-side application. For the local in-memory blockchain, we used Ganache that has a friendly user interface. We also used MetaMask, an add-on in Chrome, Firefox, and Opera, as the wallet to secure private keys and manage accounts. For development purposes, external accounts are imported from the ganache blockchain in metamask. TensorFlow JS is used to build and train machine learning models in the browser or nodejs. The main implementation of IPFS is *go-ipfs*. It includes an IPFS core implementation, an IPFS daemon server, the extensive command-line tooling, an HTTP API for controlling the node, and an HTTP Gateway for serving content to HTTP browsers.

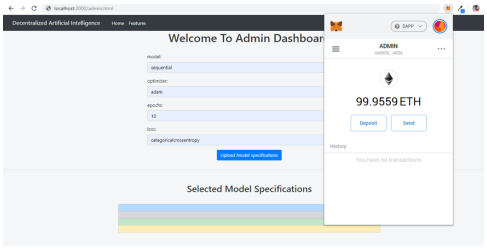

Fig. 3. Admin Dashboard

B. Implementation Setup

Initialization. A truffle project is first initialized using a truffle suite to generate some necessary files and folders required to begin the actual implementation for the project. These files and folders are as follows: *app directory* where HTML, stylesheets and JavaScript files reside, *contracts directory* to include all necessary smart contracts for the application, *migrations directory* for migration files. Whenever we deploy smart contracts, we are changing the state of the blockchain and therefore need a migration. *Truffle.js* file is the configuration file for the truffle project and also used by the ganache blockchain.

Client and Administrator Implementation. Client-side is developed using HTML, CSS, and JavaScript. It includes *TensorFlow.js* integration to train AI, neural network models, locally. Moreover, it has *web3.js* and *ipfs-http-client* to establish a connection to the Ganache blockchain and store and retrieve the AI model in a distributed file system, respectively. The administrator side is developed similar to the client but does not include AI model training capability.

Smart Contract Implementation for Admin and Client. Smart contracts are programmed in *solidity* language. Smart contracts developed in the system use *struct* and *mapping* data structures to store the client and admin details such as account address, name, etc. Mapping can be viewed as hash tables that contain key and map to its value. Constructor and private functions are used to initialize values at the time of smart contract deployment. In the system, constructors can set default model specifications. Public functions are also used to develop specific functionalities that can be used by the clients and administrators depending on the access control to these functionalities. The Truffle project needs to be configured in Ganache properly. After the configuration is done, smart contracts can be deployed. In addition, a connection between client browser and Ganache need to be established and configured using MetaMask.

C. Interfaces

Admin Interface. An administrator is a person who can set specifications for a neural network. These specifications are the parameters to configure the neural network layers. Admin can also update and query previously set specifications. Fig. 3 shows the web interface for the administrator dashboard.

Client Interface. Clients interact with the system using the web interface. Clients who are data contributors are provided

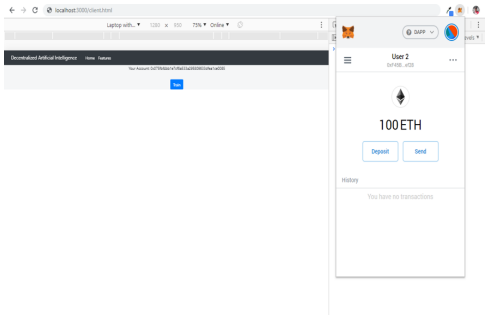

Fig. 4. Client Dashboard

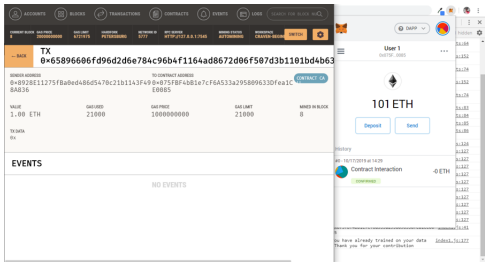

Fig. 5. Client Reward Transaction

with a "Train" button to train a neural network on its local dataset. Both actors are external accounts and need to login through meta mask extension to interact with the Ethereum network. Fig. 4 shows the web interface for the client.

The system configures a neural network with the admin specifications stored in a smart contract. When the client clicks on the train button, the system creates a web3 connection to check if the logged-in client has previously trained on its data. The system will allow the client to train on its data *only once* when the client has not participated in the training process. A message like you have already trained on your data" will be prompted if client has already participated in training. When a client allows the system to access its local data for model training, the client gets rewarded for its contribution from admin in ether. In our prototype, one ether is transferred for clients' contributions from the administrator, as shown in Fig. 5. In practice, this amount is significant for single training on a small amount of data and, therefore, is set for only testing purpose. The amount awarded in reality can be adjustable.

The system uses *tenserflow.js* to configure and compile a neural network model using model specifications from a smart contract. When the first client completes the model training, it is stored in IPFS. Moreover, its hash is stored in a smart contract, as discussed in the overview of the system. When clients click on the train button, the system initially checks for any available model stored in IPFS by checking the presence of hash stored by any previous client in the smart contract. The model is updated sequentially by distributed clients. The administrator can access this model at any point in time and uses it to perform predictions. This process does not reveal any data and identity of clients.

VI. Experiments and Results

A. Experiments Setup

Dataset. The dataset used for this system is the Iris flower data [16]. It consists of samples of each of the species of the Iris flower (Iris *setosa*, Iris *virginica*, and Iris *versicolor*). The dataset contains 130 records having five attributes: *sepal length*, *sepal width*, *petal length*, *petal height* (features), and *species* (label). This dataset is used for training by clients, i.e., data contributors, and a subset of the data is distributed among multiple clients who participate in training the model. Our experiments use two clients for illustration. Please note that we only use the dataset to demonstrate the use of blockchain for federated learning, so the complexity of dataset and specific federated learning algorithms are not the focus of this study.

TensorFlow Neural network Specifications. In our experiments, we train on the Iris flower dataset to build a neural network model. This model typically consists of AI network functions called parameters that help the model to learn from data and to tune the model itself. In our prototype, the administrator is in charge of setting these AI model parameters. For this purpose, a specific set of parameters is used to analyze the model performance, such as the number of dense layers, activation functions, optimizers, and loss. Table I lists an example of parameters used to conduct the experiments.

Experimental Settings. To understand how our prototype behaves, we conduct a series of control experiments, with the settings provided in Table II. Table III summarizes the results from the five experiments, with detailed explanations and interpretations elaborated for each of them.

B. Results

Experiment 1. This is a preliminary experiment performed to verify whether the model trained by a client can be used to train by a successor client on its data to get an updated and improved model. Every client trains the model for ten epochs. Training loss is calculated during each epoch. If there is a decrease in this value, it shows improvement in the performance of the model. We perform this experiment with two clients (Client 1 and Client 2) to check if training loss continues to decrease with distributed training.

From the above results Fig. 6 and Fig. 7, the model is trained by Client 1, and training loss calculated at last epoch is approximately 0.6311. Client 2 uses the same model to perform training on its dataset and starts with the training loss of 0.5956 and ends at 0.4325. We see that the model improves gradually, and reusing the model on a distributed dataset does not cause a training process to re-initialize the model. Training time required for Client 1 and Client 2 in Experiment 1 is 9.766 sec and 9.11 sec, respectively.

Experiment 2. In this experiment, we equally split the dataset into two training sets. These training sets are distributed among two clients. Client 1 has the first half of the dataset, and Client 2 has the second half, i.e., each containing 65 training samples. We investigated the performance of the model by observing the training loss achieved with experiment

TABLE I
ADMINISTRATOR MODEL SPECIFICATIONS

Model	No. of Dense Layers	Activation (Dense Layer)	Activation (Output Layer)	optimizer	loss
TensorFlow Classification	2	sigmoid	softmax	adam	categorical Crossentropy

TABLE II
EXPERIMENT SETTINGS

Experiment No.	Dataset	Epochs	Client 1	Client 2	Model Used
1	Full	10	✓	×	Initial
	Full	10	×	✓	From Client 1
2	First 50%	10	✓	×	Initial
Test 1	Last 50%	10	×	✓	From Client 1
3	First 50%	10	✓	×	Initial
Test 2	Last 50%	10	×	✓	From Client 1
4	First 50%	20	✓	×	Initial
	Last 50%	20	×	✓	From Client 1
5	Full	20	✓	×	Initial

TABLE III
SUMMARY OF EXPERIMENT RESULTS

	Client 1 Training Loss	Time taken for Training (in seconds) by Client 1	Client 2 Training Loss	Time taken for Training (in seconds) by Client 2
Experiment 1	0.6311	9.76	0.4325	9.11
Experiment 2	0.7657	6.24	0.7748	6.34
Experiment 3	0.5629	6.28	0.7064	6.23
Experiment 4	0.3187	32.88	0.3682	15.83
Experiment 5	0.3421	27.17	×	×

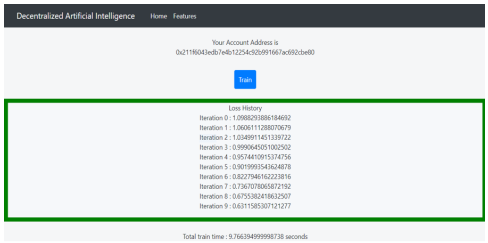

Fig. 6. Client 1 Training in Experiment 1

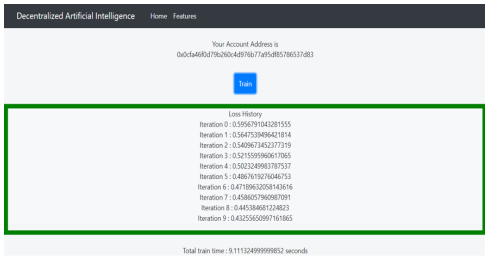

Fig. 7. Client 2 Training in Experiment 1

2 settings. Fig. 8 shows the training loss achieved at each epoch. We observe that the neural network model in the first epoch did not learn the relation between features and labels from the dataset to classify the flower species correctly and returned 1.144 loss. Similarly, we could see that in the subsequent epochs, the training loss reduces, and client 1 achieved 0.765 loss after 10 epochs. The experiment also includes the training performed at client 2. As Client 2 received a model from client 1, and the model is only trained with Client 1s data. It has not been fed with Client 2s data yet. Therefore, the model tries to learn from new data for the next ten epochs. In Fig. 9, we see that training loss for the first epoch has increased, i.e., 1.630, as the model is fed with the data for the first time. Successive epochs in Fig. 9 shows a model improves as the loss reduces and achieved 0.774 at the last epoch. This is less than that result obtained by Client 2 in Experiment 1. This is because, in Experiment 1, both clients train on the same training set and model learned for 20 epochs on the entire dataset combined and provided excellent results. To verify if the Experiment 2 data distribution is the cause of this result, we use the same data distribution and settings and perform a similar test in Experiment 3 and observe the results. Training time required for Client 1 and Client 2 in Experiment 2 is 6.247 sec and 6.347 sec, respectively, which is less than the time taken by each individual client in Experiment 1.

Experiment 3. Experiment 3 is conducted to confirm the model behavior observed in Experiment 2. It will confirm if the results obtained in the above experiment are due to the data distribution or the models internal working based on the model specification parameters such as Adam optimizer. The results show Client 1 training loss on the first epoch, which is 0.806, as opposed to the 1.144 in Experiment 2. Differences in the loss can be observed due to the optimizer used, and its internal working. In this experiment, we achieved 0.562 and 0.706 loss for Client 1 and Client 2, respectively. Comparing these results with Experiment 2, there is no significant difference in the

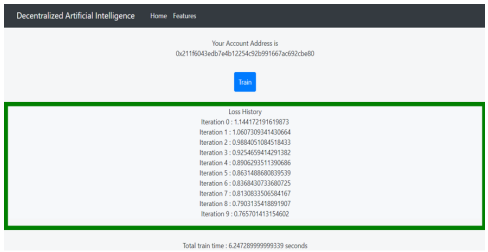

Fig. 8. Client 1 Training in Experiment 2 (first 50% data)

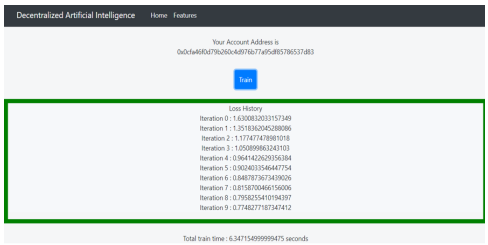

Fig. 9. Client 2 Training in Experiment 2 (last 50% data)

result. The final loss achieved by the model in Experiment 2 was 0.774 and showed similar behavior in this experiment. Training time required for Client 1 and Client 2 for this test is 6.285 sec and 6.231 sec, respectively.

Experiment 4. For this experiment, we consider similar data distribution as in Experiments 2 and 3, but there is a difference in the epoch settings. In this experiment, we have increased the number of epochs to allow the model to learn more from the features of the data keeping the model specification the same. We found significant improvement in the model training loss. The final loss observed on Client 2 reached 0.368. We observe that the training loss achieved is better than the loss achieved in Experiment 1. Training time recorded for Client 1 and Client 2 is 32.886 sec and 15.831 sec, respectively, which is more than the training time observed in Experiment 1.

Experiment 5. This experiment is conducted in a way similar to the traditional approach to simulate the model training behavior of the centralized system to learn from the complete dataset. There is no data distribution among clients as there is only one client who has a complete dataset. Training is performed only once by Client 1 for 20 epochs. Training loss begins with 1.17 for the first epoch, similar to the above experiments. The final training loss observed in the last epoch is 0.3421.

Due to space constraints, the clients' training loss results are omitted. They can be found in the detailed report [17].

VII. Conclusion and Future Work

We proposed an approach to train the AI model in a decentralized way by using Ethereum blockchain. We built a prototype system to demonstrate how the approach can be used and how effective it is. We tested our system using an Iris flower dataset and successfully train the AI model on clients distributed datasets. The results show that the model trained has similar performance while training the AI model in a serverless environment compared with the traditional centralized AI training system. However, the time required for computation is nearly double that of traditional systems due to the number of epochs needed to achieve the required performance. Our reward system allows clients to get incentivized for their data contribution without compromising data privacy, data control, and data security. Moreover, the final optimized model can further be used by the administrator to provide better services to its customers.

The future work for this paper can be as follows: 1) The current system can be used as a base system to develop solutions for problems in domains like healthcare, e-commerce, etc., and a more extensive evaluation can be conducted in those scenarios; 2) Data contributors may add fake data for model training to earn incentives. To avoid such training, data contributors' data must be validated before allowing any contributors/clients to train on their data.

Acknowledgement

Xiaoyan Sun and Jun Dai are supported by NSF DGE-2105801.

References

[1] Ray Kurzweil. "The Age of Intelligent Machines." MIT Press, 1990.
[2] Argility. "Artificial Intelligence: Machine Learning: Deep Learning." https://www.argility.com/argility-ecosystem-solutions/iot/machine-learning-deep-learning/.
[3] Brendan McMahan, H., Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. "Communication-efficient learning of deep networks from decentralized data." arXiv e-prints (2016): arXiv-1602.
[4] Ma, Chuan, Jun Li, Ming Ding, Howard H. Yang, Feng Shu, Tony QS Quek, and H. Vincent Poor. "On safeguarding privacy and security in the framework of federated learning." IEEE network 34, no. 4 (2020): 242-248.
[5] Nguyen, Dinh C., et al. "Federated learning meets blockchain in edge computing: Opportunities and challenges." IEEE Internet of Things Journal 8.16 (2021): 12806-12825.
[6] Li, Yuzheng, et al. "A blockchain-based decentralized federated learning framework with committee consensus." IEEE Network 35.1 (2020): 234-241.
[7] Zhilin Wang, and Qin Hu. "Blockchain-based federated learning: A comprehensive survey." arXiv preprint arXiv:2110.02182 (2021).
[8] Google AI Blog, April 6, 2017. "Federated Learning: Collaborative Machine Learning without Centralized Training Data. https://ai.googleblog.com/2017/04/federated-learning-collaborative.html.
[9] Qiang Yang, Yang Liu, Tianjian Chen, Yongxin Tong. "Federated Machine Learning". ACM Transactions on Intelligent Systems and Technology, 10(2), 1-19, 2019.
[10] Zahra Ghodsi, Tianyu Gu, and Siddharth Garg. "Safetynets: Verifiable execution of deep neural networks on an untrusted cloud". In Advances in Neural Information Processing Systems, pages 46724681, 2017.
[11] A. Trask Ryffel, M. Dahl, B. Wagner, J. Mancuso, D. Rueckert, and J. Passerat-Palmbach. "A generic framework for privacy preserving deep learning". http://arxiv.org/abs/1811.04017. 2018.
[12] Wood, et. al. "Ethereum: A secure decentralised generalised transaction ledger". Ethereum project yellow paper, 151(2014):132, 2014.
[13] Ethereum.org. "Learn about Ethereum. https://ethereum.org/learn/#smart-contracts.
[14] What Is IPFS? "IPFS Documentation." https://docs.ipfs.io/introduction/overview/.
[15] IPFS Distributions. https://dist.ipfs.io/#go-ipfs.
[16] Fisher, R. A. "The use of multiple measurements in taxonomic problems". Annals of Eugenics, 7, Part II, 179188. The data were collected by Anderson, Edgar (1935). https://www.kaggle.com/rtatman/iris-dataset-json-version.
[17] Anand Surendra Masurkar. "Decentralized Artificial Intelligence with Data Privacy Protection." https://hdl.handle.net/10211.3/215217.