DFL: High-Performance Blockchain-Based Federated Learning

Tian, Yongding ; Guo, Zhuoran ; Zhang, Jiaxuan ; Al-Ars, Zaid

DOI
10.1145/3600225

Publication date
2023

Document Version
Final published version

Published in
Distributed Ledger Technologies: Research and Practice

DFL: High-Performance Blockchain-Based Federated Learning

YONGDING TIAN, ZHUORAN GUO, JIAXUAN ZHANG, and ZAID AL-ARS, Delft University of Technology, Netherlands

Many researchers have proposed replacing the aggregation server in federated learning with a blockchain system to improve privacy, robustness, and scalability. In this approach, clients would upload their updated models to the blockchain ledger and use a smart contract to perform model averaging. However, the significant delay and limited computational capabilities of blockchain systems make it inefficient to support machine learning applications on the blockchain.

In this article, we propose a new public blockchain architecture called DFL, which is specially optimized for distributed federated machine learning. Our architecture inherits the merits of traditional blockchain systems while achieving low latency and low resource consumption by waiving global consensus. To evaluate the performance and robustness of our architecture, we implemented a prototype and tested it on a physical four-node network, and also developed a simulator to simulate larger networks and more complex situations. Our experiments show that the DFL architecture can reach over 90% accuracy for non-I.I.D. datasets, even in the presence of model poisoning attacks, while ensuring that the blockchain part consumes less than 5% of hardware resources.

CCS Concepts: • **Computing methodologies** → **Machine learning approaches**; *Distributed computing methodologies*; • **Computer systems organization** → **Peer-to-peer architectures;**

Additional Key Words and Phrases: Federated machine learning, blockchain, partial consensus, reputation

ACM Reference format:
Yongding Tian, Zhuoran Guo, Jiaxuan Zhang, and Zaid Al-Ars. 2023. DFL: High-Performance Blockchain-Based Federated Learning. *Distrib. Ledger Technol.* 2, 3, Article 20 (September 2023), 25 pages.
https://doi.org/10.1145/3600225

1 INTRODUCTION

In the past ten years, people have witnessed the rise of crypto-currency and the growth of **decentralized finance** (**DeFi**). Most current blockchain applications are limited to the financial domain because it is non-sensitive to performance limitations but relies on data security and integrity. [12] and [20] show some potential blockchain application areas such as **Internet of Things** (**IoT**) and health care. However, only limited practical applications exist in these areas due to the low **transaction per second** (**TPS**) and high latency of blockchain systems.

Federated learning is a **machine learning** (**ML**) training method where multiple clients send their local model updates to a central server, which will later generate a new global model and broadcast it to clients. Federated learning addresses privacy issues by keeping the training data on clients [45], but it raises trust challenges

This research was performed with the support of the European Commission, the ECSEL JU project New Control (grant no. 826653) and the Eureka PENTA project Vivaldy (grant no. PENT191004).
Authors' address: Y. Tian, Z. Guo, J. Zhang, and Z. Al-Ars, Delft University of Technology, Delft, South Holland 2600 AA, Netherlands; emails: {Y.Tian-14, Z.Guo-4, J.Zhang-42}@student.tudelft.nl, z.al-ars@tudelft.nl.

2769-6472/2023/09-ART20 $15.00
https://doi.org/10.1145/3600225

between clients and central servers. For example, clients can send incorrect models to the central server to harm the global model. Servers can also infer the training data on a specific client using the changes in its model, causing privacy leakage [3, 8].

To address the above challenges, some researchers suggest utilizing blockchain technology as a replacement for central servers [2, 6, 26, 36]. This is possible because blockchain offers a transparent and honest environment for generating and broadcasting the new global model. Blockchain also offers a reliable record of the models submitted by clients, thus malicious clients can be identified and removed. Mainstream research in federated learning on blockchain uses smart contracts to perform model aggregation, which results in poor performance because generating a block usually costs over 10 seconds [18, 32].

This article proposes a new blockchain architecture turning the blockchain database from a distributed ledger into a distributed proof of contribution to the ML model, where each node can generate its own blocks asynchronously, significantly improving the overall federated learning efficiency. The new architecture also shows reasonable stability and robustness, with a suitable node reputation strategy and weighted FedAvg implementation for non-I.I.D. datasets and model poising attacks. From the ML perspective, the main contribution of this article is an asynchronous Gossip-alike ML training method with blockchain that stores proof of contribution to the ML models. We implement this method in the DFL framework, including an executable prototype, a simulator to simulate the ML behavior, and an SDK to design reputation/model updating algorithms. This article also mentions other algorithms, such as the reputation algorithm and weighted federated averaging algorithm, but they are not the main contributions because they are used to study the characteristics of the DFL framework.

This article will first discuss current federated ML and blockchain systems in Section 2. Section 3 provides a high-level overview of DFL, as well as its application area and its differences with federated ML. The detailed elaboration, including data format, blockchain structure, and federated ML implementation, is presented in Section 4. Section 5 shows the blockchain performance and federated ML performance. The conclusions and future works are presented in Section 6.

2 RELATED WORK AND BACKGROUND

2.1 Blockchain System and Bifurcation

Blockchain systems were first introduced in bitcoin [29], a peer-to-peer, public, and immutable ledger system. All transactions in bitcoin are sealed into blocks to achieve immutability. These blocks are then stored in a directed chain where each block contains the digest of the previous block. The directed chain ensures that any changes in a block will result in the change of all following blocks. Additionally, a global consensus algorithm is applied to ensure that only one chain exists through all nodes in this peer-to-peer network. Bitcoin uses **proof of work** (**PoW**) to reach the global consensus, where bitcoin miners must "dig" for a random number to make the block digest meet a certain pattern. The difficulty of finding such a random number varies with the computation power in the whole blockchain network - it becomes more difficult to find a qualified random number with more computational power in total.

In a blockchain system, the computation power of the network is measured by the speed of generating blocks, or equivalently, the interval between two adjacent blocks. This interval time is important for the blockchain system as it determines the probabilities of bifurcation. The term bifurcation describes a situation where in a blockchain network, two nodes generate two correct blocks simultaneously. Since each node only accepts the first block received, one of the blocks would be rejected by over half of all miners. All miners will try to generate the next block based on their current received block. In bifurcation, one block is more likely to generate the next block once it is accepted by most miners because more miners mean more computation resources to find the next block. So as time goes by, one chain will be much longer than the other chains and be accepted by all miners, where the global consensus is reached again. The bifurcation can significantly degrade the performance

of the blockchain system and should be avoided. One standard solution is increasing the mining difficulty, which leads to a longer block generation interval. Meanwhile, a longer interval also brings higher latency. However, The bifurcation problem exists in blockchain systems no matter the consensus algorithms.

2.2 Federated Machine Learning (Federated ML)

Federated ML continues to draw increasing attention from the research community. A typical federated ML system contains a central server and many clients. The central server is responsible for aggregating and updating the models trained by clients, which are then sent back to clients for the next training iterations. This method of aggregating and updating the central model is called FedAvg [27]. Federate ML has advantages in both data privacy since the training data is privately kept to the client, and model training quality achieves high accuracy for I.I.D., non-I.I.D., and unbalanced datasets [11].

However, federated ML also brings new vulnerabilities to the system, including data poisoning [1, 43], model poisoning [9], and privacy attack from the server [35]. Data poisoning and model poisoning aim at providing wrong models to the central server to harm the training process of the global model. Besides, the central server is able to recover the protected dataset stored on the client by applying **Generative Adversarial Networks (GANs)** [7, 35].

Even though federated ML decentralizes ML systems from central servers to client devices, the central server must be kept to perform model aggregation and updating. This makes the central server a single point of failure to the entire system. To deal with this single point of failure and potential attacks on privacy, solutions were proposed where the central server can be substituted by a blockchain [16, 18, 32].

2.3 Federated ML on Blockchain

Bitcoin, as the first blockchain system, has limited functionality as it only provides a distributed and immutable ledger. The next-generation blockchain systems, such as Ethereum [40], provide smart contracts to allow users to execute specific small applications on the blockchain system. The smart contract is more reliable because its source code is open to the public on the blockchain and all miners will execute it to ensure the integrity of the results. Some researchers hence attempt to integrate federated ML with Ethereum and replace the model aggregating server with smart contracts [18, 32]. However, applying smart contracts on public blockchain systems is expensive due to the miner fee. Federated ML on blockchains would also be slow because every data aggregation and computation will be performed on all nodes. A brief cost analysis is available in [32], where federated ML costs over 10^8 Ethereum gas - a cost that would be even greater for larger and more complex ML tasks. This is also the reason why [18] uses a private blockchain system to avoid high costs. In addition to the above problems, the storage also becomes expensive for federated ML because every model from all ML clients will be stored on all blockchain nodes.

Based on the previous discussion, here is a summary of the problems of current ML approaches to blockchain systems.

- Low blockchain performance: limited transactions per block and high latency resulting from long block generation interval.
- Mining cost: ML applications must afford the miner fee, which is an extra expense for ML applications.
- Excessive storage cost: ML models will be stored on all blockchain miner nodes because the models are stored in a transaction, which will be packed into a block. All miners keep the blockchain database so every single model will be stored on each miner's disk.

The above three problems hinder the integration of federated ML on the blockchain, even though there are already many theoretical works to improve the performance of federated ML on the blockchain. For example, [39] proposes a compression method to reduce network communication during training; [23] focuses on privacy issues in industrial IoT applications. However, only a few of them are built on a real blockchain system [18, 32].

In this article, we propose a practical framework (DFL) with (1) a deployable executable, (2) a simulator to simulate ML performance, and (3) an SDK to support further research.

3 HIGH-LEVEL OVERVIEW OF DFL

DFL is a **peer-to-peer** (**p2p**) network where users collaborate for model training by exchanging model parameters. The models are updated asynchronously through (i) training when sufficient local data become available to the nodes and (ii) averaging when a given number of models are received. DFL applies a local ledger to register the training contribution of each node to be used as an incentive mechanism. DFL implements partial consensus which is more effective in this case, because partial consensus allows each node to do model averaging when they receive models rather than waiting for a block to be finalized. DFL introduces a distributed reputation system to mitigate dataset and model attacks as partial consensus makes attackers hiding easier compared with traditional blockchain-based FL solutions.

Section 3.1 talks about the partial consensus and its prototype - gossip algorithm in ML. Section 3.2 elaborates on the distributed reputation system. Section 3.3 and Section 3.4 show the potential applications of DFL and compare DFL with existing related projects.

3.1 Gossip Algorithm and Partial Consensus

Gossip algorithms have been a popular choice for decentralized federated learning research, as they enable the sharing of models in P2P networks [13, 17, 33]. DFL also utilizes a gossip mechanism, in which each node only shares its models with a limited number of neighbors and invites them to be witnesses of its contribution to the ML models. While gossip algorithms offer a decentralized approach to model sharing, they may not provide the same level of transparency and trust as a blockchain-based system because a single node doesn't obtain the information of the whole network.

From a blockchain perspective, the gossip mechanism used in DFL is called "partial consensus", as the ML model (transaction) is only broadcast to a limited range of nodes, rather than the entire blockchain network as in global consensus. Partial consensus is suitable for ML applications for two main reasons. Firstly, the order in which transactions arrive is not critical, as a transaction that arrives late can still be used in the current averaging round. Secondly, the model weights contained in the transactions are not critical. Two models with the same accuracy, as determined by the same test dataset, may have different model weights.

As mentioned in Section 2.1, global consensus in blockchain systems can lead to bifurcation, and many contemporary blockchain systems increase the block interval to avoid this issue. In contrast, the partial consensus in DFL relaxes the strict synchronization requirement and thus allows for improved model communication efficiency and reduced training time. The role of blockchain databases also shifts from an information-sharing system to proof of contribution for the ML models. Each node generates and maintains its own blockchain database, which records feedback from its neighbors (i.e., witnesses). However, partial consensus can also introduce new vulnerabilities, such as the potential for malicious behaviors like model poisoning to go unnoticed in the network. To address these concerns, a reputation system will be introduced in the next section to incentivize honest participation and mitigate the risk of malicious behavior.

3.2 Distributed Reputation

There are already some existing reputation systems in the federated ML area. For example, [15] proposed using reputation as an incentive mechanism. The reputation system in DFL is responsible for distinguishing the malicious node in partial consensus. A malicious node can be a node frequently broadcasting low-accuracy models, or performing a data poisoning attack, or performing a model poisoning attack. The reputation is distributed and different nodes may rate the same node independently. For example, considering a three-node system - the reputation of node C in node A might be different from that stored in node B.

The reputation can be updated according to any rational ML model criterion. In this article, we mainly consider updating the reputation according to the model's accuracy. More specifically, each node will have its own local reputation record of its neighbors and will update the reputation record based on the accuracy of neighbors' models calculated by itself based on its local dataset. Broadcasting models with higher accuracy earn the node a higher reputation in the system. The reputation will be stored locally and not shared with other nodes.

To protect the system, a malicious node should have a lower reputation in the reputation system. Meanwhile, the influence of malicious models should be lowered as well. We propose using the weighted federated averaging method [41], a modified version of FedAvg, in this article. The modified FedAvg mechanism works together with the reputation system to keep the federated ML secured and stable.

In Section 5, we implemented a sample reputation and weighted average system and obtained promising results. However, there are many other factors that can affect the implementation of this system, such as the malicious rate, training performance, and data distribution. In future work, we plan to investigate the impact of these factors on the reputation and weighted FedAvg system and develop more universal implementations that can handle a wide range of variations.

3.3 Possible Applications

This subsection talks about potential applications for DFL. These applications are only feasible with proper reputation algorithms and federated averaging algorithms, which are not covered in this article.

One potential application is replacing current blockchain-based federated ML systems for less training time and lower resource consumption. Meanwhile, the blockchain database can serve as an incentive mechanism and allow further reward distribution, avoiding the miner fee issue in traditional blockchain-based federated ML systems.

Another possible application is allowing independent developers to deploy their ML models without running expensive cloud servers or paying miner fees. For example, they can embed DFL in their apps to collect data and train models. The reward can be replaced with in-app benefits to compensate users for hardware resources.

DFL also makes it possible to build community-driven ML ecosystems, in which developers can propose ML models that are trained using data contributed by users and advertise them to attract miners to collaborate on the training process. Smart contracts on traditional blockchain systems can be used to receive DFL blockchain data and distribute token rewards to miners as an incentive for their participation. These tokens can then be used for ML inference in the future if the model is adequately trained, and people can buy the tokens from miners to pay the inference fee. As a result, ML models that are more useful and have higher accuracy are expected to have higher token values on the traditional blockchain, reflecting their perceived value in the ecosystem.

3.4 Comparison

[12] summarizes existing blockchain-based federated ML systems and identifies four major issues that are addressed by these systems: (1) single-point failure, (2) poison attack, (3) lack of motivation, and (4) privacy leakage. Table 1 lists the addressed issues by each system analyzed in [12] and makes a comparison with DFL. The reasons for each addressed issue in DFL are listed below:

— Single point failure: there is no central server or central ML model, so there is no single point of failure.
— Privacy leakage: the model in each transaction is a mixture of model training from the dataset and model updating with transactions from other nodes. In addition, each transaction will only be broadcast to limited other nodes due to the ttl mechanism. These two features make privacy attacks much harder.
— Mining fee: the DFL network avoids mining fees by allowing each node to generate its own blockchain data, unlike traditional blockchain-based federated ML systems that rely on blockchain systems and require mining fees for executing smart contracts.

Table 1. Comparison with other Blockchain-based Federated ML Systems

System	Addressed issues	Type of blockchain	Consensus mechanism
BlockFL [16]	Single point failure, lack of motivation	Public	PoW
FLChain [2]	Single point failure, lack of motivation	Public	–
FLChain [26]	Single point failure	Public	PBFT/PoW
RFL [15]	Poison attack, lack of motivation	Consortium	PBFT
DeepChain [38]	Single point failure, lack of motivation, privacy leakage	Public	Blockwise-BA
FedBC [42]	Single point failure, privacy leakage	–	–
BC-FL [24]	Single point failure, lack of motivation	Public	Pow
BlockFLA [6]	Single point failure, poison attack, lack of motivation	Public & private	PoW & PBFT
Chain FL [18]	Single point failure	Private	PoA
PSFL [44]	Poison attack	–	–
BFEL [14]	Single point failure, poison attack	Public & consortium	DPoS/PBFT & PoV
Biscotti [34]	Single point failure, poison attack, privacy leakage	Public	PoF
VFChain [31]	Single point failure, poison attack	Public	Algorand
BLADE-FL [21]	Single point failure	Public	PoW
BFLC [22]	Single point failure, poison attack, lack of motivation	Public	Committee
VBFL [4]	Single point failure, poison attack, lack of motivation	Public	PoS
DFL	Single point failure, privacy leakage, mining fee, synchronicity in blockchain, lack of motivation (with a proper rewarding algorithm), poison attack (with a proper reputation mechanism)	Public	Partial consensus

— Synchronicity in the blockchain: DFL is an asynchronous system because it does not require global consensus among the nodes in the network. This asynchronicity can reduce the strict synchronization requirements of traditional blockchain systems to allow for faster training of the ML models.
— Lack of motivation: the blockchain data on that node proves the node's contribution to ML models. The contribution is quantified as improvements in accuracy measured by neighbors. The final reward should be calculated by an algorithm (not covered in this article) involving the blockchain data and the reputations of that node.
— Poison attack: while the design of a reputation algorithm to prevent poison attacks is not covered in this article, DFL provides an SDK for further research on this topic. As shown in Section 5, dataset poison attacks are less effective in DFL because most nodes are innocent. Model poison attacks require additional research.

There are several projects that are similar to DFL but are not based on blockchain technology. For example, IPLS [30] is a decentralized federated learning system that uses the **inter-planetary file system** (**IPFS**) for model dissemination. In terms of federated learning, IPLS applies the segmented model to save bandwidth, which is also different from DFL. Another project, BrainTorrent [33], is a peer-to-peer federated learning system that is similar to DFL in terms of the federated learning process. However, BrainTorrent requires nodes to actively choose their peers and collect models, whereas DFL uses a broadcast mechanism for model dissemination. Overall, these projects demonstrate that there are alternative approaches to federated learning that can be implemented without the use of blockchain technology.

4 TECHNICAL DESCRIPTION OF DFL

Figure 1 shows a high-level overview of the DFL workflow. The workflow consists of two event-driven phases. The left phase is triggered when the node has collected enough "ML training data" for training (e.g., when the training data size becomes equal to/larger than the ML batch size). The node will create a "transaction" that includes the new "ML model", and broadcast it to other nodes. After several rounds of communications, the node

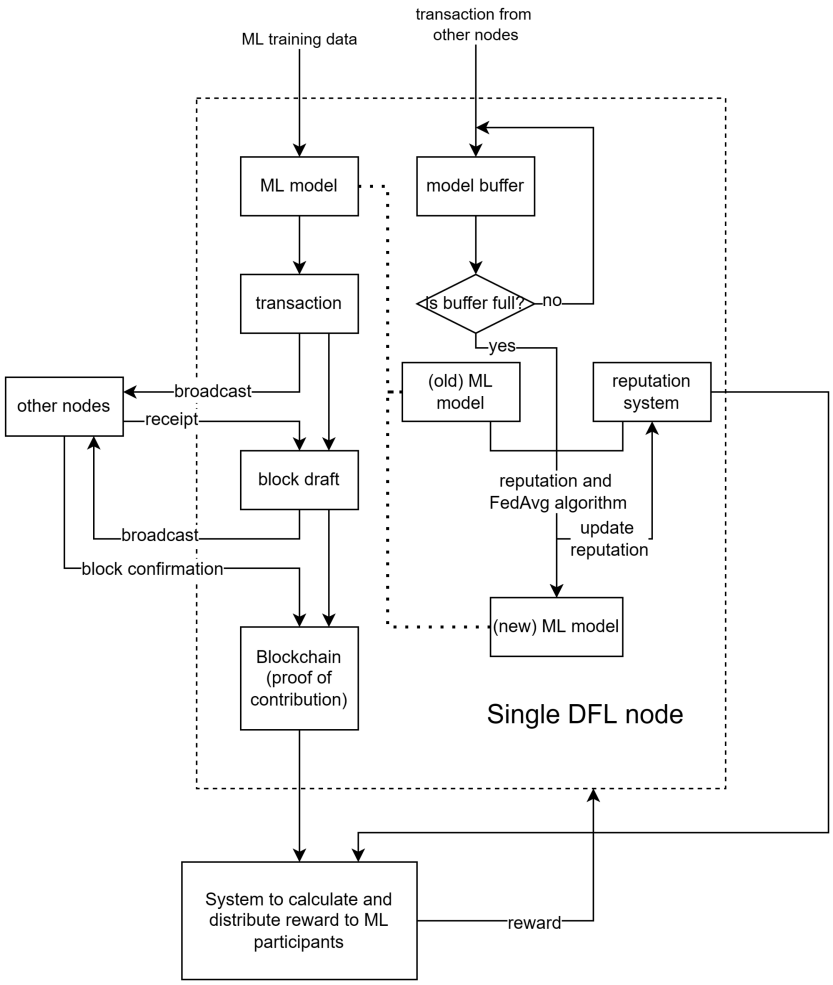

Fig. 1. DFL overview.

can generate the "blockchain (proof of contribution)". The right phase is triggered when the node has received enough transactions (controlled by "model buffer" size) from other nodes to update the ML model and ends by updating the "(new) ML model" and "reputation system".

The blockchain data serves as proof of contribution and can be submitted to a dedicated reward calculation system to quantify the contributions to ML models and distribute the rewards to the DFL nodes. This could be an incentive mechanism to encourage each DFL node to train and share their models.

4.1 DFL Workflow

In this section, a fully-connected four-node DFL system will be used as an example to explain the workflows and structures. Please keep in mind that DFL does not rely on a fully-connected network to operate, less p2p connections also ensure the system's correctness.

Table 2 lists some important concepts which will be used in this section. In the table, either one of the mainstream asymmetric encryption methods, such as ECDSA and RSA, could be applied in the system.

Table 2. Notion Table for Section 4.1

Notion	Definition
pri_key_{node}	The private key for a node.
pub_key_{node}	The public key for a node.
$address_{node}$	An unique identifier for a node.
$hash(content)$	The hash for certain content. The content will be immutable once *hash* is calculated.
$signature(pri\_key, hash)$	The signature generated by the pri_key and *hash*.
$generator$	The node of generating transactions and blocks.
$neighbors\ of\ x$	All nodes whose network distance to node x is 1.
$further\ neighbors\ of\ x$	All nodes whose network distance to node x is larger than 1.

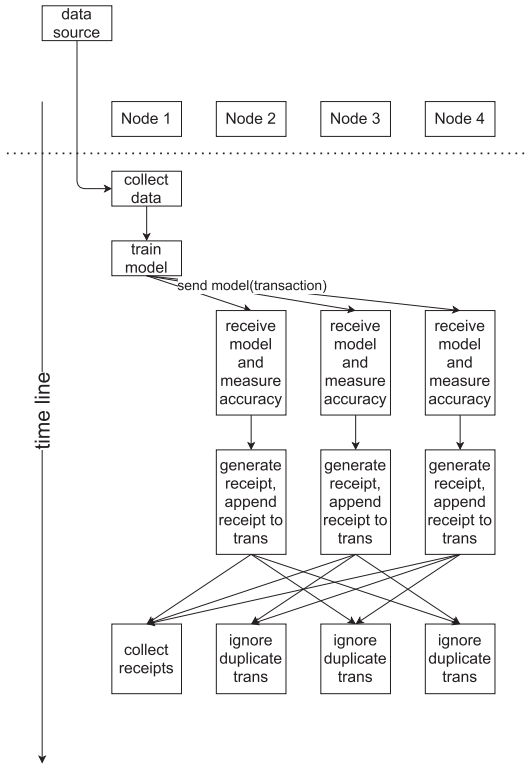

Fig. 2. Procedures of generating transaction and receipt.

4.1.1 Node Setup. Each node in the DFL network generates a pair of public/private keys and an address to join the DFL network. The key pair can be generated by most cryptology methods, while the address is produced according to the rule $address_{node} = hash(pub\_key_{node})$.

4.1.2 Generating Transaction and Receipt. Figure 2 describes the procedures to generate transactions.

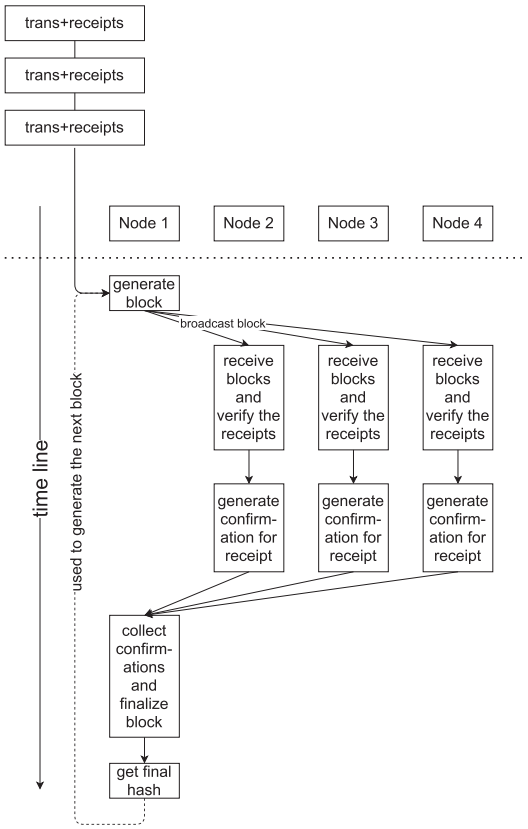

Fig. 3. Procedures of generating a block.

Suppose in our DFL system, node 1 is training at the beginning. The model weights after training are sent to the neighbors of node 1, i.e., nodes 2, 3, and 4 since the network is fully connected. Once other nodes receive the weights, they will start to measure the model accuracy based on their own collected dataset. Each receiver will generate a receipt describing the accuracy of the model, which is then appended at the end of the transaction. The transaction, together with the receipt, will be sent from each receiver to all its neighbors. For instance, node 2 as receiver of model weights from node 1, will send the transaction and receipt to, not only node 1 but also nodes 3 and 4 as well. However, nodes 3 and 4 will ignore the transaction and receipt from node 2, as this transaction has been processed beforehand. Node 1 will collect the receipt for generating the block afterward.

4.1.3 Generate Block. Figure 3 describes the procedures to generate blocks. Node 1 will gather a certain amount of transactions and corresponding receipts after a certain time. A block would be generated to record the transactions and receipts in order to make them immutable.

The procedure for generating such a block can be divided into two phases. The block generator creates a draft block in the first phase. The second phase is to gather confirmations from neighbors of the block generator and finalize a block. With confirmations appended to the final block, the block generator cannot easily modify the previous block as it cannot fake a confirmation.

Retake the four-node system as an example. Node 1 sends a draft block to node 2, then node 2 will look through the receipts to figure out whether some receipts are produced by node 2 and generate confirmations if there are

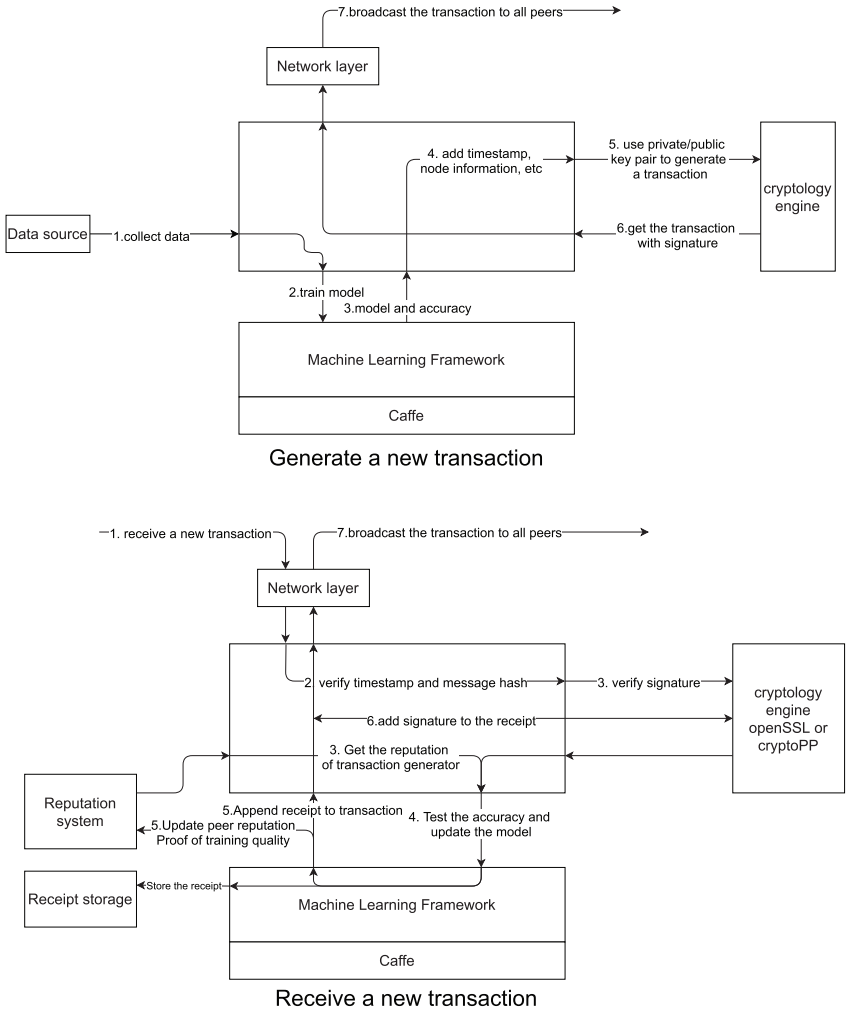

Fig. 4. Generating transaction (block diagram).

any. Node 1 will gather all confirmations from nodes 2, 3, and 4 and append the confirmations to finalize the block. The digest of the final block will then be used to generate the next block.

The component diagram is available in Figures 4 and 5. The number before each step indicates the order. These steps present the same process flow as Figures 2 and 3, so they will not be discussed again.

4.2 DFL Blockchain Data format (UML Available in Figure 6)

In this section, the data formats in the blockchain system will be discussed based on the workflow in previous sections. To be specific, the data formats include the format of the transaction, the transaction receipt, the block, and the block confirmation. These data formats will be used to prove the robustness of the DFL blockchain system.

These data formats will be illustrated in Table 3 with some special notions.

4.2.1 Node Information. The name of a node is its address. Since the node address is decoupled from the network address, it is hard to access a specific node by its name. This property could achieve similar anonymity as

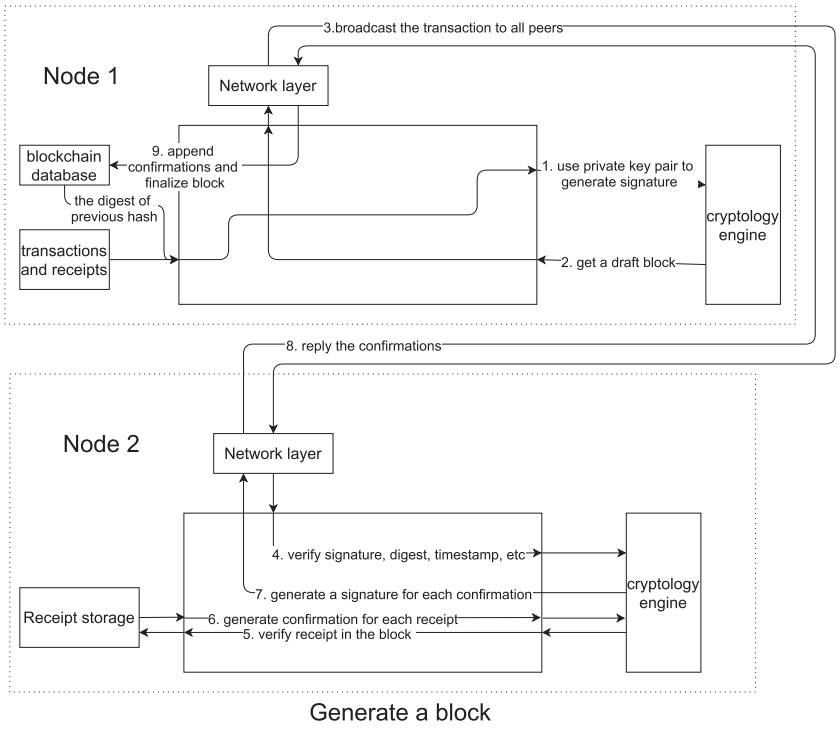

Fig. 5. Generating block (block diagram).

Table 3. Notion Table for Section 4.2

Notion	Definition
= (*operation*)	This data is obtained by performing the operation.
hash (*content*)	The hash for a certain content, the content will be immutable once *hash* is calculated.
signature (*pri_key*, *hash*)	The signature generated by the *pri_key* and *hash*.
digest protected (*digest d*)	This data is protected by the digest d. If the original data is modified, the digest verification will fail.
signature protected (signature s, node o)	This data is definitely generated by *node o* and has not been modified once the signature verification passes. Because a signature is generated based on a digest, all signatures-protected data is also digest-protected.

the blockchain since nodes can only receive a trusted transaction from the node but hard to access the generator. For neighbors, they can know each other's network address because they will receive a transaction with no receipt.

4.2.2 Transaction Format. Among all the transaction fields, the *generator*, *create_time*, *expire_time*, *ml_model*, and *ttl* are protected by the signature, which can only be modified by the transaction creator. A modified transaction with a changed digest will be treated as a new transaction.

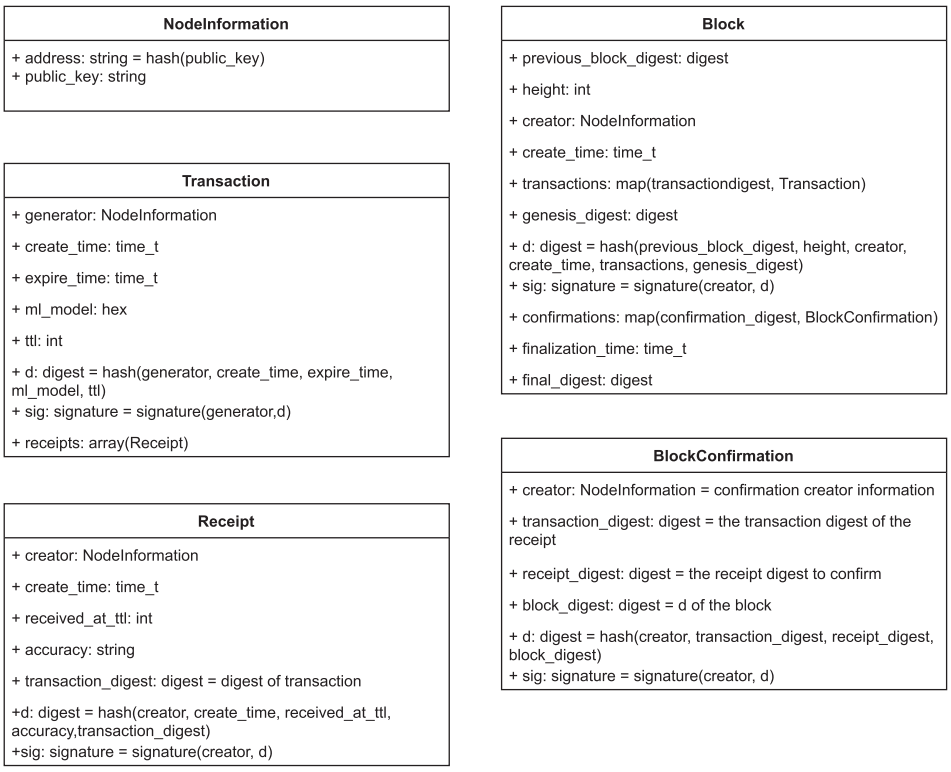

Fig. 6. UML graph for DFL blockchain system.

The *ttl* (time to live) is an integer value decreased by 1 when the transaction is forwarded by a node. The transaction will cease being forwarded if its *ttl* reaches zero. The *ttl* value is critical for the range of partial consensus. Larger *ttl* results in a wider partial consensus range with more nodes involved in the accuracy testing for this transaction. So an infinite *ttl* should be avoided because it might consume too much computing resource. The mechanism to punish a node generating large-*ttl* transactions can be achieved in the reputation system.

The *expire_time* is designed to avoid a late arrived transaction, whose ML model has already been outdated, i.e., processed by other nodes. Those outdated ML models can harm the overall training process and reduce the creator's reputation. This expiration time can as well avoid attackers' attempts to decrease the reputation of the victims by delaying their transactions.

4.2.3 Receipt Format. The *creator* is the receipt creator, corresponding to nodes 2,3, and 4 in the four nodes example in Figure 2. The *transaction_digest* is the digest of the received transaction. The transaction digest will not change after appending new receipts because the transaction digest does not take *receipts* into the computation. The *received_at_ttl* is calculated by the following equation.

$$received\_at\_ttl = min(trans.ttl, trans.receipts.received\_at\_ttl) - 1. \tag{1}$$

The *accuracy* is tested based on the receipt creator's dataset. In DFL, all nodes will not shuffle their dataset, so a model can have different accuracy for different nodes, especially for non-I.I.D. dataset. The accuracy will be forwarded back to the transaction creator and other nodes. This accuracy information might be helpful for other nodes because it assists to judge the receipt creator's reputation, but it has not yet been implemented in the current DFL version.

4.2.4 Block Format. The $transactions$ in the block are the final collected transactions which contain receipts. Neighbors should verify the correctness of the receipts and generate a block confirmation after verification. The block generator should collect a certain number of confirmations from other nodes in the network in order to finalize the block. The $genesis\_digest$ is the digest of the genesis block, which records the ML network structure to ensure all nodes use the same ML network. So all nodes in DFL should have the same $genesis\_digest$ if they are training the same model. The $final\_digest$ is calculated after the block generator collects enough confirmations for each transaction, and will be chained to the next block.

4.3 DFL Blockchain Properties

4.3.1 Asynchronous. One of the key features of DFL is its ability to operate asynchronously, as it does not rely on global consensus. This means that the generating transaction, block, receipt, and confirmation processes on different nodes can be performed independently of each other. Additionally, the ML process and the generating blockchain process can also be asynchronous, as long as it is not necessary to record the contribution in the current or next block. This asynchronous nature of DFL allows nodes to operate independently, which can enhance the scalability and efficiency of the system.

4.3.2 Anonymity. All nodes are anonymous on the DFL network, as their IP addresses are hidden behind their node addresses and their data packets are only available to their direct peers. While this anonymity provides some level of protection for nodes, it is important to note that the node address itself may serve as an identifier for an attacker who wants to recover the dataset held by a victim. There are already several blockchain projects, such as Monero [28], that aim at enhancing node anonymity. However, this article will not further discuss the issue of anonymizing node addresses.

4.3.3 Immutability. In DFL, blocks become immutable once they are finalized. This is because neighbors will not generate block confirmations for a receipt more than once, and the block generator cannot modify any content in the receipts or confirmations, as they are protected by the signatures of other nodes. The immutability of blocks in DFL helps to ensure that the contribution to the ML models cannot be modified. This is important for maintaining the integrity of the system and ensuring that all nodes are properly incentivized for their participation.

4.4 Discussion

Here are some discussions and comments about the DFL system:

- The DFL blockchain data can be deleted to save storage after the contributors have received their rewards. Suppose the reward system for DFL is a smart contract on a traditional blockchain. It is important to minimize the amount of data that is stored on the traditional blockchain, so we should keep the on-chain data minimized to reward algorithm arguments, such as model accuracies.
- In DFL, the communication cost depends on the ML model size, ttl, and the number of peers. In traditional blockchain-based federated ML, the communication cost depends on the model size (or gradient size) and the communications in the blockchain system. So the practical communication cost is application-specific and difficult to compare.
- Server-side attacks are not applicable because there are no centralized servers. Peer-side attacks, such as model poisoning and dataset poisoning, are a potential concern in DFL.
- Privacy attacks are harder to perform in DFL than in a federated learning system because the ML models will only be shared with a limited number of nodes in the network. The ML models in the blockchain data can also be deleted before submitting to the rewarding system if the rewarding algorithm does not require the ML models.
- As for network failures, failure to broadcast transactions can cause some transactions to be missed, resulting in fewer contributions to the network and fewer blocks. Failure to send draft blocks results in fewer

blockchain confirmations. In practical DFL deployment, each node only needs to collect a certain percentage of confirmations in order to finalize a block. Disconnected nodes can catch up with the rest of the network after re-connection by using a model updating mechanism that gives higher weights to more accurate machine learning models.

— An attack method to fake contributions is possible in DFL. This attack method can be described as an attacker that steals the model from other legitimate nodes and generates a new transaction that includes the copied model to gain contributions. Verifying the digest of a transaction does not help because the attacker can slightly modify the model weights to get a completely different digest while retaining similar accuracy. There are two potential ways to defend against this attack. One is to use **Trusted Execution Environments** (**TEEs**) [37] on all nodes to protect the model data. The other way is to use homomorphic encryption [25] to encrypt the model, which allows other nodes to use the model without decrypting it.

5 RESULT AND DISCUSSION

This section uses experiments to measure the DFL performance and simulations to investigate the ML properties regarding malicious nodes and non-I.I.D. datasets.[1] We also upload the simulation configuration files and reputation binary files to make the simulation results re-producible. The ML model used in our experiment and simulations is LeNet [19], and the dataset is MNIST [5]. We use the same hyperparameters as the LeNet example in Caffe [10] because improving the ML models is not our focus. We implement two types of reputation and weighted FedAvg methods. The first implementation is a slightly modified version of federated averaging without reputation. The following formula describes the model updating procedure on each node:

$$model_{next} = \frac{\sum_{n=1}^{N} \frac{1}{N} \cdot model_n + model_{prev}}{2}. \tag{2}$$

The $model_{next}$ is the output model of this model updating iteration, $model_n$ represents the received models from other nodes, $model_{prev}$ is the local output model of the last model updating iteration. N represents the total number of models in the model buffer. This formula does not utilize the reputation mechanism, and we call this formula "`HalfFedAvg`" in this article because, in this formula, each node's own model contributes to 50 percent of the new model, while models received from other nodes together contribute the other 50 percent.

The second implementation is called "`reputation-0.05`", as described in the following formula:

$$weight = reputation \cdot accuracy, \tag{3}$$

$$model_{next} = \frac{\sum_{n=1}^{N} \frac{weight_n}{\sum_{m=1}^{N} weight_m} \cdot model_n + model_{prev}}{2}, \tag{4}$$

$reputation$ is the reputation of the model generator, ranging from 0 to 1. The node with the lowest accuracy will be punished by reducing its reputation by 0.05 in each iteration (the punishment will stop if the reputation is already 0). $accuracy$ stands for the calculated accuracy from the model in the transaction, ranging from 0 to 1. Suppose there are N models in the FedAvg buffer, the total weight is obtained by summing all weights (like in the denominator $weight_m$), and the previous model is marked as $model_{prev}$. A special case in "`reputation-0.05`" is that the reputations of all peers become 0. In this case, the mechanism will be replaced by "`HalfFedAvg`".

We note here that the algorithms and mechanisms introduced above are not the main contributions of this article. The main contribution of this article is the DFL framework itself, and the above algorithms are used to show the DFL framework is possible to be applied in malicious and non-I.I.D. cases.

[1] DFL source code available: https://github.com/twoentartian/DFL/tree/9d8d0a2a4e5e05e91460d005a567a8c73b608739.

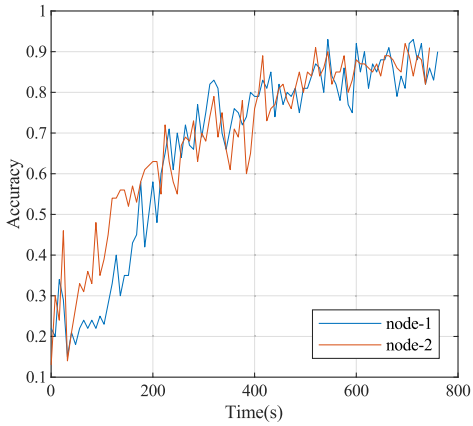

Fig. 7. Accuracy graph for 2 nodes.

5.1 DFL Performance (Experiment)

The goal of this experiment is to investigate the performance of the DFL network. The built-in performance profiler in the DFL executable can measure the CPU time consumed by each task in the DFL components (generate block, generate transactions, update models, and receive transactions). The GPU acceleration in Caffe is disabled to ensure the metric of CPU time is fair for ML and blockchain tasks. We provide two DFL testnets, a four-node net, and a two-node net. Both testnets use the "`reputation-0.05`" mechanism.

5.1.1 The Two-node Net. The two-node test net is configured as each node will receive eight randomly-selected data samples from the MNIST training dataset per second. The training batch size is 64, so each node will perform a training every 8 seconds in theory. A transaction will be generated and broadcast immediately after the training. Each node will generate a block after broadcasting 4 transactions, equivalent to 32 seconds. The test batch size is 100, and each node will randomly choose 100 samples from all received samples. We do not use the test dataset here because choosing from received samples is closer to practical cases. On a higher level, other DFL nodes with different received samples can replace the role of the test dataset. The FedAvg buffer size is set to 4, causing the FedAvg and reputation mechanism to run roughly every 32 seconds.

Figure 7 shows the accuracy graph. The accuracy is tested based on local received data on each node. It causes a rapid increase (from 0s to 30s) in accuracy at the beginning of training. The rapid drop near 32 seconds is caused by the FedAvg process, which merges the models from other nodes with different datasets. Table 4 shows the performance result and Table 5 shows the blockchain statistics. The performance result indicates that the blockchain part (blockchain overhead) only consumes a small portion of CPU time, the network (broadcast transaction, gather confirmation, and broadcast generated transaction) consumes much more, and ML (calculate accuracy, calculate self-accuracy, and measure accuracy) costs the most. In DFL, the blockchain data is proof of contribution to ML models, and spending less time generating such proof is always desirable.

Notice that the time in Table 4 is CPU time while the time in Figure 7 (x-axis) is wall clock time. Because DFL is a multi-threading application, summing all time in Table 4 will result in a larger value than wall clock time.

5.1.2 The Four-node Net. Some configuration modifications are applied to the four-node DFL network: the training sample received by each node is decreased from 8 to 4 per second; the FedAvg buffer size is increased from 4 to 8; the transaction count in each block is increased from 4 to 12. The network topology is fully-connect. The results are shown in Figure 8, Table 6, and Table 7.

Table 4. Two Node Performance Result, Unit: CPU Second

node-1	Generate block (s)		Generate transaction (s)		
	Gather confirmation	Blockchain overhead	Broadcast transaction	Measure accuracy	Blockchain overhead
	21.52	3.48	59.10	362.83	18.31
	Update model (s)		Receive transactions (s)		
	Calculate self-accuracy	Blockchain overhead	Broadcast generated transaction	Calculate accuracy	Blockchain overhead
	82.94	0.94	80.57	341.73	5.50
node-2	Generate block (s)		Generate transaction (s)		
	Gather confirmation	Blockchain overhead	Broadcast transaction	Measure accuracy	Blockchain overhead
	22.22	2.40	72.81	312.30	7.18
	Update model (s)		Receive transactions (s)		
	Calculate self-accuracy	Blockchain overhead	Broadcast generated transaction	Calculate accuracy	Blockchain overhead
	77.16	0.43	44.89	310.63	3.62

Table 5. Two Node Blockchain Statistics

	Transaction/Block	Confirmation/Block	Peers	Block
node-1	4	4	1	24
node-2	4	4	1	24

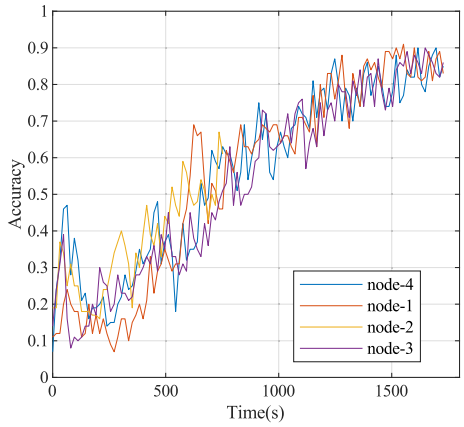

Fig. 8. Accuracy graph for 4 nodes.

In theory, the `Confirmation/Block` value should be 36 because each transaction has three confirmations, and there are 12 transactions in a block. The real value is slightly smaller because some confirmations are not collected in time. In practical deployment, the rule of finalizing a block could be "collecting at least 80 percent of all confirmations" or "collecting at least 60 percent of confirmations for each transaction" to make trade-offs on the blockchain robustness and performance.

Table 6. Four Node Performance Result, Unit: CPU Second

node-1	Generate block (s)		Generate transaction (s)		
	Gather confirmation	Blockchain overhead	Broadcast transaction	Measure accuracy	Blockchain overhead
	56.63	3.17	71.72	364.72	8.28
	Update model (s)		Receive transactions (s)		
	Calculate self-accuracy	Blockchain overhead	Broadcast generated transaction	Calculate accuracy	Blockchain overhead
	90.34	0.74	214.48	1944.49	23.33
node-2	Generate block (s)		Generate transaction (s)		
	Gather confirmation	Blockchain overhead	Broadcast transaction	Measure accuracy	Blockchain overhead
	105.66	4.00	88.89	399.76	8.86
	Update model (s)		Receive transactions (s)		
	Calculate self-accuracy	Blockchain overhead	Broadcast generated transaction	Calculate accuracy	Blockchain overhead
	99.67	3.45	299.28	1438.01	20.07

Table 7. Four Node Blockchain Statistics

	Transaction/Block	Confirmation/Block	Peers	Block
node-1	12	35.11	3	9
node-2	12	35.67	3	9

Table 8. Time Comparison: DFL vs. Baffel

	DFL	BAFFLE
Model	LeNet	2-layer DNN, 500 perceptrons each layer
Number of nodes	2∼4	16∼128
ML-related activities	2,000s∼2,500s (CPU time)	70s∼90s (wallclock time)
Blockchain-related activities	∼200s (CPU time)	2,000s (chunk size[a] 16 kB) - 11,000s (chunk size 2 kB) (wallclock time)

[a]chunk size is a blockchain parameter in BAFFLE [32].

As shown in Table 6, most CPU resources are consumed by calculating the accuracy of received models, and this time consumption is directly proportional to the number of direct neighbors. Given that the number of neighbors is restricted to a specific maximum and that the workflow on each node is asynchronous from that of other nodes, DFL does not suffer from blockchain-related scalability issues as the number of nodes in the network increases.

To illustrate that DFL has higher performance compared to other blockchain-based FL systems, we compare it to Baffle [32], a federated learning system based on a private Ethereum network with Proof-Of-Authority consensus. The results of this comparison are shown in Table 8, which indicates that the time consumed by the DFL blockchain part is 10 times smaller than Baffle. It is important to note here that the blockchain performance in Baffle is affected by the "chunk size" parameter (which is not an issue for DFL). Additionally, the reason why we do not use wall-clock time for DFL in Table 8 is that it is determined by the manually-set training data injection rate.

Despite the conclusion that the blockchain part only consumes little CPU resources, the result shows another conclusion that the CPU time consumed by calculating accuracy for models from other nodes is proportional to the number of peers. In our experiments, calculating accuracy is a bottleneck for performance because it delays replying to receipts. Re-performing the experiment with GPU accelerators may solve this bottleneck. In real deployments, the number of peers should be controlled to save hardware resources.

5.2 ML Properties (Simulation)

In this section, an investigation is conducted to assess the effect of non-I.I.D. datasets and malicious nodes on machine learning. Due to some limitations in deploying larger DFL networks, we use a simulator to replace the actual deployment. The DFL simulator makes some simplifications compared with physical DFL deployments:

— The DFL simulator removes the blockchain part.
— In DFL deployments, each node is asynchronous because they run on different computers. The simulator is a single process, and its behavior is synchronous, so we use a virtual time scale, called "Tick", to simulate the asynchronous behavior. For example, each node receives a batch of training data every "x" ticks, the "x" is a random value between 8 to 12.
— The DFL simulator does not simulate delay. The delay includes network delay and the time spent on ML. For example, a node will know the accuracy of a model from other nodes immediately once broadcasted.
— In simulator, the accuracy values are always calculated based on the MNIST test dataset rather than based on the actual received data samples received by each node.

We investigate three non-I.I.D. types of dataset. The first type is that each node only gets a dataset with two specific labels. For example, node-0 only gets training data labeled with 0 and 1; node-1 only gets training data labeled with 1 and 2. For the other two types of non-IID samples, we sampled a probability distribution over the class labels for each node based on a symmetric Dirichlet prior. In particular, we investigated the cases `alpha=1` and `alpha=0.1` where `alpha` is a concentration parameter of the Dirichlet distribution. Smaller `alpha` results in a more non-I.I.D. dataset. For non-I.I.D. cases described by the Dirichlet distribution, we will regenerate the probabilities of each label and repeat the simulation 10 times to avoid extreme results.

There are two types of malicious nodes: dataset-poisoning attackers and model-poisoning attackers. Dataset attackers always use random samples (all pixels are uniform random numbers from 0 to 1) to train models and send them to other nodes. Model attackers train the model normally but always send randomly generated models to other nodes. The weights of the model are uniform random numbers from 0 to 0.001. We choose 0.001 as the maximum weight value to avoid NaN (floating-number overflow) error in Caffe. The output of the random model is also random because of the last Softmax layer.

The DFL simulation network contains 10 nodes, and each node actively connects to two other peers, so in theory, each node will have 4 peers if no overlapping occurs. The network topology is randomly generated by a DFL tool, and all simulations in this section use the same network topology. The FedAvg buffer size is 4 regardless of peer count. The training batch size is 64, and the test batch size is 100. Each node will get a batch of training data at the training tick, which is determined as the current training tick plus a random number ranging from 8 to 12. The transaction *ttl* (defined in Figure 6) is fixed to 1.

The DFL simulator provides two metrics. The first metric is the model accuracy of each node, and the second metric is a value to describe how different the models are. The following formula calculates the value:

$$difference = \frac{\sum_{n=1}^{N-1} |model_n - model_{n-1}| + |model_0 - model_n|}{N}. \tag{5}$$

The N represents the number of nodes,[2] $model_n$ indicates the sum of the weights for a specific layer of model n. $model_n$ is a vector of 4 elements if the ML model has 4 layers. The operator $|\cdot|$ calculates the absolute value

[2]also represents the number of models because each node has one model.

Table 9. The Subplot Location for Figure 9

	IID	non-I.I.D. alpha = 1 (10 repetitions)	non-I.I.D. alpha = 0.1 (10 repetitions)	non-I.I.D. 1node2labels
no malicious HalfFedAvg	Subplot 0	Subplot 1	Subplot 2	Subplot 3
dataset poisoning HalfFedAvg	Subplot 4	Subplot 5	Subplot 6	Subplot 7
model poisoning HalfFedAvg	Subplot 8	Subplot 9	Subplot 10	Subplot 11
no malicious Reputation-0.05	Subplot 12	Subplot 13	Subplot 14	Subplot 15
dataset poisoning Reputation-0.05	Subplot 16	Subplot 17	Subplot 18	Subplot 19
model poisoning Reputation-0.05	Subplot 20	Subplot 21	Subplot 22	Subplot 23

of the vector elements. The output *difference* is a vector where each element represents the difference of a layer among N models.

As mentioned previously, there are two reputation and FedAvg mechanisms and both of them will be used in the simulation. The first mechanism is "`HalfFedAvg`" described in Equation (2) and "`Reputation-0.05`" described in Equation (4).

Because the DFL network is a new computing architecture and we observed many unexpected behaviors and phenomena that never exist in normal federated ML, we present the accuracy and weight difference results from all combinations of non-I.I.D. cases and malicious cases[3] in Figure 9. Table 9 summarizes all combinations of non-I.I.D. and malicious cases and corresponding subplot locations in Figure 9. For example, subplot 13 is located at column 2 and row 4 in Table 9, "(column 2, row 4)" also represents the subplot location in Figure 9. Each subplot has two parts: part a is the accuracy graph, and part b is the model difference among all nodes.

Figure 9 shows that the accuracy of ML models can reach over 0.9 under non-I.I.D. and dataset poisoning cases after 4,000 ticks. The "epoch" concept is not suitable for DFL because all training samples are randomly selected from the training dataset and injected to the DFL nodes. Randomly choosing training data is closer to practice because each node has no information about the dataset of other nodes in a decentralized network. The total data samples trained by the DFL network (10 DFL nodes) is $\frac{4000 \cdot 64 \cdot 10}{10} = 256{,}000$ at 4,000 ticks, approximately 4.2 epoch for MNIST training dataset. The accuracy variances in the second and third columns seem to be smaller than those in the fourth column because the accuracy plot in the second and third columns is the averaged result from ten repetitions.

Subplot 8 and Subplot 20 show that `Reputation-0.05` mechanism can prevent model poisoning in I.I.D. cases, but it does not work in non-I.I.D. cases as shown in Subplots 21,22,23. The factors for successfully training models in non-I.I.D. and model poisoning situations include reputation algorithm, network topology, and the dataset distribution of each node. We have not dived too deep into non-I.I.D. and malicious cases because the reputation

[3]the configuration file of each combination is available in https://github.com/twoentartian/DFL/tree/main/paper_result/non-iid-malicious-10nodes. Notice that for non-I.I.D. cases described by the Dirichlet distribution, there are 10 simulation output folders, and each of them contains a configuration file and a plot of accuracy and weight difference. The subplot in Figure 9 in this paper is the average of all plots in the corresponding output folders.

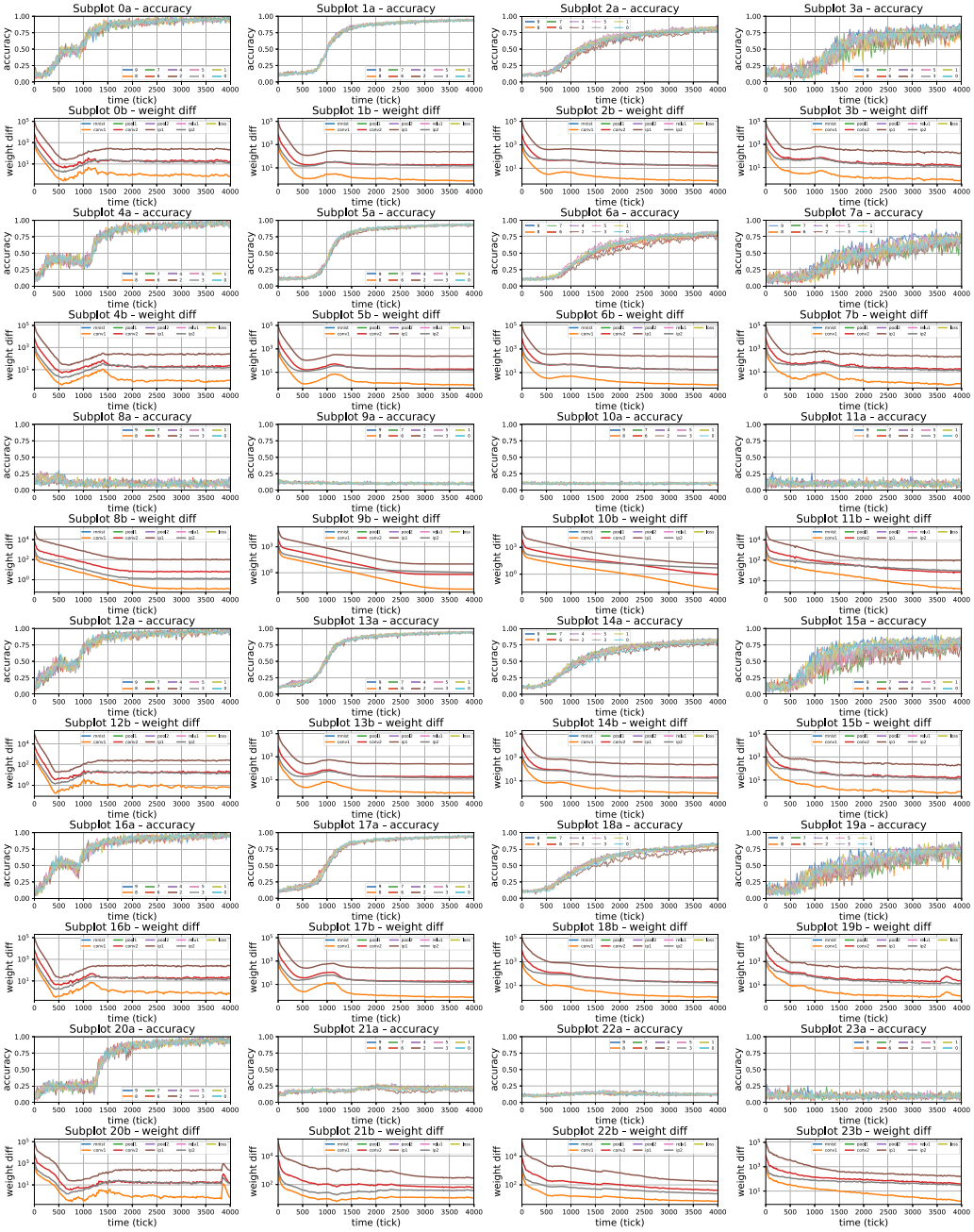

Fig. 9. Simulation results for malicious and non-I.I.D. cases (vectorized image, zoom-in for details).

mechanism is not our focus. The DFL framework provides a reputation SDK to facilitate further research on reputation algorithms.

There are some other phenomena in these subplots, which we will briefly discuss in this article. These phenomena can lead to future research topics:

— There is only one model in traditional ML systems and federated ML systems, indicating the model weight difference should be zero when a model is sufficiently trained. However, the model weight difference in DFL does not converge to zero but rather converges to a constant value when the model is properly trained. Subplots 8,9,10,11 also show unsuccessful training has a lower model weight difference than successful training. In addition, the model weight difference does not constantly decrease during the training process. The weight difference at 4,000 tick is even higher than 600 tick in Subplot 0. In other words, the training result of the DFL network is a set of models rather than a single model, and each model has similar accuracy. One of the potential applications of this property is training models whose dataset relates to geographical location. For example, a disease in different continents has different disease behavior, and there are two DFL nodes on each continent. Both DFL nodes should be able to train a model that can best predict the disease based on different disease behavior. For traditional ML systems, the geographical information must be included in the training dataset, but we would not know that geographical information is critical to the model if we do not aggregate all the data.
— There is a complex relationship between model weight difference and model accuracy, which we are not yet able to describe yet. The complexity is visible in Subplots 15 and 19 which show that the dataset distribution can change the trend of the model weight difference.
— In Subplots 0, 4, 12, and 16, there are regions where the accuracy plateaus at 0.5 from 500 ticks to 1,000 ticks. The length of the plateau regions scales up with the number of nodes,[4] and it can greatly slow down the ML training when scaling to much larger DFL networks, which requires further investigation. These plateau regions are one of the reasons we avoid simulating large-scale DFL networks because they will dominate the training performance.

5.3 The Speed Ratio of Model Training and Model Updating

This section investigates the tradeoff between bandwidth requirements and ML performance. This is done using simulations to explore the impact of the ratio of model training interval over model updating interval on ML performance. The interval of model training is controlled by training batch size and data injection rate. For example, injecting 64 training samples per 10 ticks with a training batch size of 64 results in one training iteration per 10 ticks. The interval of model updating is controlled by FedAvg buffer size, the number of peers, and the training interval of peers.

A new node type called "observer" is used in this simulation. The observer node does not train ML models and only performs FedAvg on received models. The observer can ensure that there are enough communications among nodes and that the final model is a combination of training and updating rather than an independently trained model.

Some modifications [5] are made to the configuration in Section 5.2: (1) simulations use a fully-connected 10-node DFL network; (2) each node uses an I.I.D. dataset; (3) the first node is configured as an observer; (4) the reputation algorithm is "`Reputation-0.05`". Because the training interval is kept as 8 to 12 ticks, and each node should get 8 models in approximately 10 ticks, we set the FedAvg buffer size to 32, 8, and 2 to achieve different update/train ratios (1:4, 1:1, and 4:1). The result is shown in Figure 10.

Figure 10 indicates that more communication among nodes can accelerate the training. It is a tradeoff between training speed and bandwidth because more communication requires more bandwidth consumption. We also test several extreme situations where the update/train ratio is 1:32 and the dataset distributions are I.I.D., alpha = 1, and alpha = 0.1, as shown in Figure 11. The third subplot shows that the accuracy has already increased but needs more communication. Figure 11 shows it is possible to train a model with low-frequency communications, such as in embedded systems and mobile systems.

[4]this result is based on simulations not shown in this paper.
[5]the configuration files are available in https://github.com/twoentartian/DFL/tree/main/paper_result/update_train_ratio.

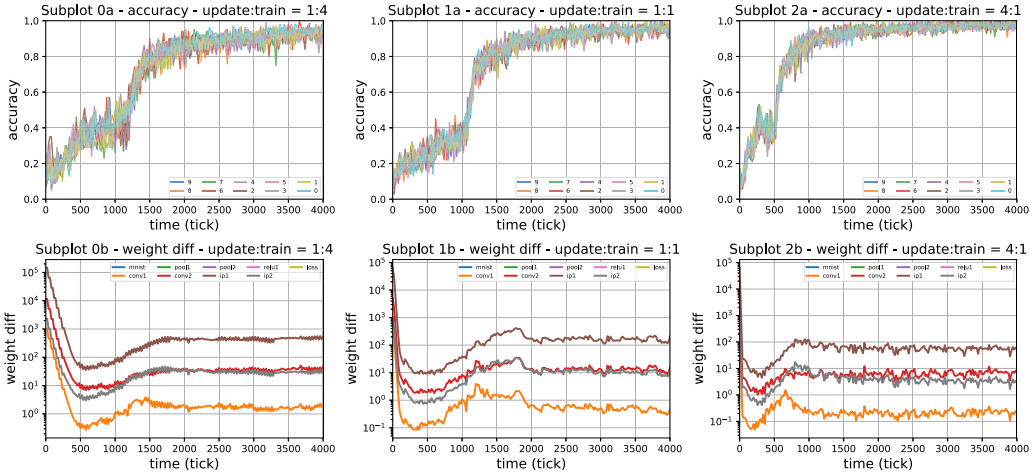

Fig. 10. Update/train ratio simulation results (update/train = 1:4,1:1,4:1).

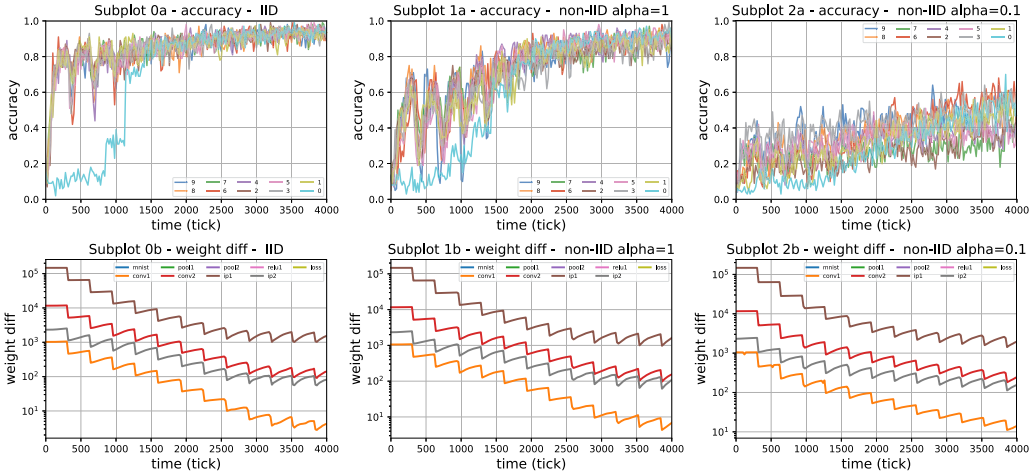

Fig. 11. Update/train=1:32 simulation results (I.I.D., alpha = 1, alpha = 0.1).

6 CONCLUSION

This article presents a new asynchronous federated learning system, integrating a specialized blockchain system to generate the proof of contribution on the ML model. Using blockchain as a distributed proof system rather than a distributed ledger system removes the synchronicity requirement to increase performance. Meanwhile, the asynchronicity also introduces the partial aggregation of models in training phase. We implement the architecture as DFL, which contains an executable prototype, a simulator to investigate ML behavior, and an SDK to implement and test reputation algorithms. In a four node experiment with MNIST and LeNet, the training accuracy can be up to 90% while the blockchain overhead can be kept limited to within only 5% of total execution time. The simulation results show reasonable training accuracy under both non-I.I.D. datasets as well as malicious dataset attacks. However, the reputation mechanism only works in certain model attack cases and requires further research.

ACKNOWLEDGMENTS

We want to thank Xun Gui and Zixuan Xie for their discussions and ideas on blockchain and machine learning. Their deep insights in blockchain and machine learning helped improve this project. We also thank Kewei Du, Xiaoning Shi, and Li Xu for reviewing this article and providing precious feedback.

REFERENCES

[1] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. 2020. How To backdoor federated learning. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research)*, Vol. 108, (2020), PMLR, 2938–2948. Retrieved from https://proceedings.mlr.press/v108/bagdasaryan20a.html.

[2] Xianglin Bao, Cheng Su, Yan Xiong, Wenchao Huang, and Yifei Hu. 2019. FLChain: A blockchain for auditable federated learning with trust and incentive. In *Proceedings of the 2019 5th International Conference on Big Data Computing and Communications*. 151–159. DOI: https://doi.org/10.1109/BIGCOM.2019.00030

[3] Abhishek Bhowmick, John Duchi, Julien Freudiger, Gaurav Kapoor, and Ryan Rogers. 2018. Protection Against Reconstruction and Its Applications in Private Federated Learning. (2018). DOI: https://doi.org/10.48550/ARXIV.1812.00984

[4] Hang Chen, Syed Ali Asif, Jihong Park, Chien-Chung Shen, and Mehdi Bennis. 2021. Robust Blockchained Federated Learning with Model Validation and Proof-of-Stake Inspired Consensus. (2021). DOI: https://doi.org/10.48550/ARXIV.2101.03300

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

[6] Harsh Bimal Desai, Mustafa Safa Ozdayi, and Murat Kantarcioglu. 2021. BlockFLA: Accountable Federated Learning via Hybrid Blockchain Architecture. In *Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy (CODASPY'21)*. Association for Computing Machinery, Virtual Event, USA, 101–112. DOI: https://doi.org/10.1145/3422337.3447837

[7] David Enthoven and Zaid Al-Ars. 2021. An Overview of federated deep learning privacy attacks and defensive strategies. 173–196. DOI: https://doi.org/10.1007/978-3-030-70604-3_8

[8] David Enthoven and Zaid Al-Ars. 2022. Fidel: Reconstructing private training samples from weight updates in federated learning. In *Proceedings of the 2022 9th International Conference on Internet of Things: Systems, Management and Security*. 1–8. DOI: https://doi.org/10.1109/IOTSMS58070.2022.10062088

[9] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. 2020. Local model poisoning attacks to byzantine-robust federated learning. In *Proceedings of the 29th USENIX Security Symposium*. USENIX Association, 1605–1622. Retrieved from https://www.usenix.org/conference/usenixsecurity20/presentation/fang.

[10] Github. 2022. github.com. (2022). Retrieved 25 May 2022 from https://github.com/BVLC/caffe/blob/master/examples/mnist/lenet_solver.prototxt.

[11] Chaoyang He, Songze Li, Jinhyun So, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, Li Shen, Peilin Zhao, Yan Kang, Yang Liu, Ramesh Raskar, Qiang Yang, Murali Annavaram, and Salman Avestimehr. 2020. FedML: A research library and benchmark for federated machine learning. arXiv:2007.13518. Retrieved from https://arxiv.org/abs/2007.13518.

[12] Dongkun Hou, Jie Zhang, Ka Lok Man, Jieming Ma, and Zitian Peng. 2021. A systematic literature review of blockchain-based federated learning: Architectures, applications and issues. In *Proceedings of the 2021 2nd Information Communication Technologies Conference*. 302–307. DOI: https://doi.org/10.1109/ICTC51749.2021.9441499

[13] Chenghao Hu, Jingyan Jiang, and Zhi Wang. 2019. Decentralized federated learning: A segmented gossip approach. arXiv:1908.07782. Retrieved from http://arxiv.org/abs/1908.07782.

[14] Jiawen Kang, Zehui Xiong, Chunxiao Jiang, Yi Liu, Song Guo, Yang Zhang, Dusit Niyato, Cyril Leung, and Chunyan Miao. 2020. Scalable and communication-efficient decentralized federated edge learning with multi-blockchain framework. In *Proceedings of the Blockchain and Trustworthy Systems*. Zibin Zheng, Hong-Ning Dai, Xiaodong Fu, and Benhui Chen (Eds.), Springer Singapore, Singapore, 152–165.

[15] Jiawen Kang, Zehui Xiong, Dusit Niyato, Shengli Xie, and Junshan Zhang. 2019. Incentive mechanism for reliable federated learning: A joint optimization approach to combining reputation and contract theory. *IEEE Internet of Things Journal* 6, 6 (2019), 10700–10714. DOI: https://doi.org/10.1109/JIOT.2019.2940820

[16] Hyesung Kim, Jihong Park, Mehdi Bennis, and Seong-Lyun Kim. 2020. Blockchained on-device federated learning. *IEEE Communications Letters* 24, 6 (2020), 1279–1283. DOI: https://doi.org/10.1109/LCOMM.2019.2921755

[17] Anastasia Koloskova, Sebastian U. Stich, and Martin Jaggi. 2019. Decentralized stochastic optimization and gossip algorithms with compressed communication. arXiv:1902.00340. Retrieved from http://arxiv.org/abs/1902.00340.

[18] Caner Korkmaz, Halil Eralp Kocas, Ahmet Uysal, Ahmed Masry, Oznur Ozkasap, and Baris Akgun. 2020. Chain FL: Decentralized federated machine learning via blockchain. In *Proceedings of the 2020 2nd International Conference on Blockchain Computing and Applications*. 140–146. DOI: https://doi.org/10.1109/BCCA50787.2020.9274451

[19] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324. DOI: https://doi.org/10.1109/5.726791

[20] Dun Li, Dezhi Han, Tien-Hsiung Weng, Zibin Zheng, Hongzhi Li, Han Liu, Arcangelo Castiglione, and Kuan-Ching Li. 2022. Blockchain for federated learning toward secure distributed machine learning systems: A systemic survey. *Soft Computing* 26, 9 (2022), 4423–4440. DOI: https://doi.org/10.1007/s00500-021-06496-5

[21] Jun Li, Yumeng Shao, Kang Wei, Ming Ding, Chuan Ma, Long Shi, Zhu Han, and H. Vincent Poor. 2022. Blockchain assisted decentralized federated learning (BLADE-FL): Performance analysis and resource allocation. *IEEE Transactions on Parallel and Distributed Systems* 33, 10 (2022), 2401–2415. DOI: https://doi.org/10.1109/TPDS.2021.3138848

[22] Yuzheng Li, Chuan Chen, Nan Liu, Huawei Huang, Zibin Zheng, and Qiang Yan. 2021. A blockchain-based decentralized federated learning framework with committee consensus. *IEEE Network* 35, 1 (2021), 234–241. DOI: https://doi.org/10.1109/MNET.011.2000263

[23] Yunlong Lu, Xiaohong Huang, Yueyue Dai, Sabita Maharjan, and Yan Zhang. 2020. Blockchain and federated learning for privacy-preserved data sharing in industrial IoT. *IEEE Transactions on Industrial Informatics* 16, 6 (2020), 4177–4186. DOI: https://doi.org/10.1109/TII.2019.2942190

[24] Chuan Ma, Jun Li, Long Shi, Ming Ding, Taotao Wang, Zhu Han, and H. Vincent Poor. 2022. When federated learning meets blockchain: A new distributed learning paradigm. *IEEE Computational Intelligence Magazine* 17, 3 (2022), 26–33. DOI: https://doi.org/10.1109/MCI.2022.3180932

[25] Zainab Hikmat Mahmood and Mahmood Khalel Ibrahem. 2018. New fully homomorphic encryption scheme based on multistage partial homomorphic encryption applied in cloud computing. In *Proceedings of the 2018 1st Annual International Conference on Information and Sciences*. 182–186. DOI: https://doi.org/10.1109/AiCIS.2018.00043

[26] Umer Majeed and Choong Seon Hong. 2019. FLchain: Federated learning via MEC-enabled blockchain network. In *Proceedings of the 2019 20th Asia-Pacific Network Operations and Management Symposium*. 1–4. DOI: https://doi.org/10.23919/APNOMS.2019.8892848

[27] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA (Proceedings of Machine Learning Research)*, Vol. 54, PMLR, 1273–1282. Retrieved from http://proceedings.mlr.press/v54/mcmahan17a.html.

[28] monero project. 2022. Monero: the secure, private, untraceable cryptocurrency. (2022). Retrieved 12 July 2022 from https://github.com/monero-project/monero.

[29] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. Retrieved from https://bitcoin.org/bitcoin.pdf.

[30] Christodoulos Pappas, Dimitris Chatzopoulos, Spyros Lalis, and Manolis Vavalis. 2021. IPLS: A framework for decentralized federated learning. In *Proceedings of the 2021 IFIP Networking Conference*. 1–6. DOI: https://doi.org/10.23919/IFIPNetworking52078.2021.9472790

[31] Zhe Peng, Jianliang Xu, Xiaowen Chu, Shang Gao, Yuan Yao, Rong Gu, and Yuzhe Tang. 2022. VFChain: Enabling verifiable and auditable federated learning via blockchain systems. *IEEE Transactions on Network Science and Engineering* 9, 1 (2022), 173–186. DOI: https://doi.org/10.1109/TNSE.2021.3050781

[32] Paritosh Ramanan and Kiyoshi Nakayama. 2020. BAFFLE : Blockchain based aggregator free federated learning. In *Proceedings of the 2020 IEEE International Conference on Blockchain*. 72–81. DOI: https://doi.org/10.1109/Blockchain50366.2020.00017

[33] Abhijit Guha Roy, Shayan Siddiqui, Sebastian Pölsterl, Nassir Navab, and Christian Wachinger. 2019. BrainTorrent: A peer-to-peer environment for decentralized federated learning. arXiv:1905.06731. Retrieved from http://arxiv.org/abs/1905.06731.

[34] Muhammad Shayan, Clement Fung, Chris J. M. Yoon, and Ivan Beschastnikh. 2021. Biscotti: A blockchain system for private and secure federated learning. *IEEE Transactions on Parallel and Distributed Systems* 32, 7 (2021), 1513–1525. DOI: https://doi.org/10.1109/TPDS.2020.3044223

[35] Mengkai Song, Zhibo Wang, Zhifei Zhang, Yang Song, Qian Wang, Ju Ren, and Hairong Qi. 2020. Analyzing user-level privacy attack against federated learning. *IEEE Journal on Selected Areas in Communications* 38, 10 (2020), 2430–2444. DOI: https://doi.org/10.1109/JSAC.2020.3000372

[36] Joost Verbraeken, Martijn de Vos, and Johan Pouwelse. 2021. Bristle: Decentralized Federated Learning in Byzantine, Non-i.i.d. Environments. (2021). arXiv:2110.11006. Retrieved from https://arxiv.org/abs/2110.11006.

[37] Yong Wang, June Li, Siyu Zhao, and Fajiang Yu. 2020. Hybridchain: A Novel Architecture for Confidentiality-Preserving and Performant Permissioned Blockchain Using Trusted Execution Environment. *IEEE Access* 8, (2020), 190652–190662. DOI: https://doi.org/10.1109/ACCESS.2020.3031889

[38] Jiasi Weng, Jian Weng, Jilian Zhang, Ming Li, Yue Zhang, and Weiqi Luo. 2021. DeepChain: Auditable and privacy-preserving deep learning with blockchain-based incentive. *IEEE Transactions on Dependable and Secure Computing* 18, 5 (2021), 2438–2455. DOI: https://doi.org/10.1109/TDSC.2019.2952332

[39] Leon Witt, Usama Zafar, KuoYeh Shen, Felix Sattler, Dan Li, and Wojciech Samek. 2021. Reward-based 1-bit compressed federated distillation on blockchain. arXiv:2106.14265. Retrieved from https://arxiv.org/abs/2106.14265.

[40] Gavin Wood. 2014. Ethereum: A secure decentralised generalised transaction ledger. Retrieved from https://ethereum.github.io/yellowpaper/paper.pdf.

[41] Hongda Wu and Ping Wang. 2021. Fast-Convergent Federated Learning With Adaptive Weighting. *IEEE Transactions on Cognitive Communications and Networking* 7, 4 (2021), 1078–1088. DOI: https://doi.org/10.1109/TCCN.2021.3084406

[42] Xin Wu, Zhi Wang, Jian Zhao, Yan Zhang, and Yu Wu. 2020. FedBC: Blockchain-based decentralized federated learning. In *Proceedings of the 2020 IEEE International Conference on Artificial Intelligence and Computer Applications*. 217–221. DOI: https://doi.org/10.1109/ICAICA50127.2020.9182705

[43] Jiale Zhang, Bing Chen, Xiang Cheng, Huynh Thi Thanh Binh, and Shui Yu. 2021. PoisonGAN: Generative poisoning attacks against federated learning in edge computing systems. *IEEE Internet of Things Journal* 8, 5 (2021), 3310–3322. DOI: https://doi.org/10.1109/JIOT.2020.3023126

[44] Keshan Zhang, Huawei Huang, Song Guo, and Xiaocong Zhou. 2020. Blockchain-based participant selection for federated learning. In *Proceedings of the Blockchain and Trustworthy Systems*. Zibin Zheng, Hong-Ning Dai, Xiaodong Fu, and Benhui Chen (Eds.), Springer Singapore, Singapore, 112–125.

[45] Stefan Zwaard, Henk-Jan Boele, Hani Alers, Christos Strydis, Casey Lew-Williams, and Zaid Al-Ars. 2020. Privacy-preserving object detection & localization using distributed machine learning: A case study of infant eyeblink conditioning. CoRR abs/2010.07259, (2020). Retrieved from https://arxiv.org/abs/2010.07259

Received 29 November 2021; revised 23 January 2023; accepted 11 May 2023