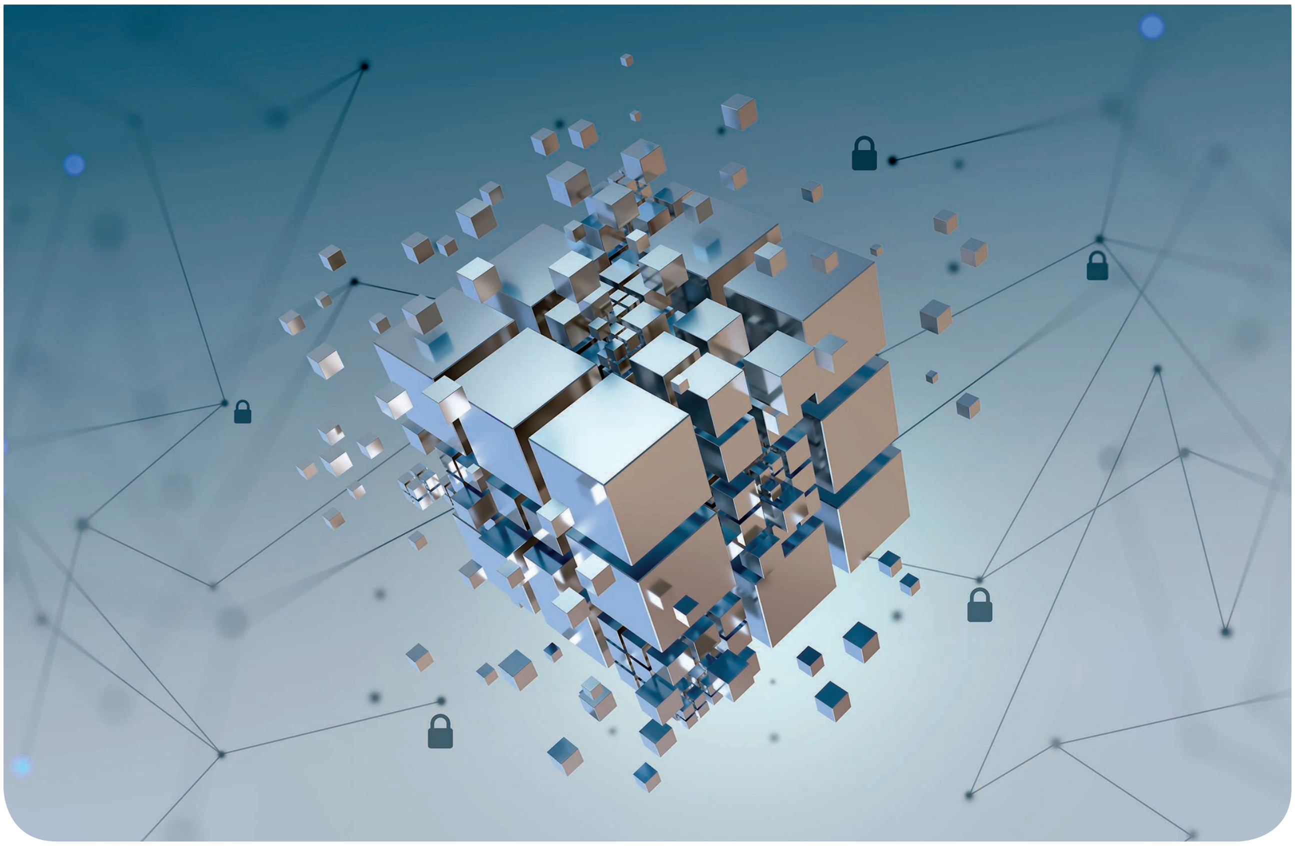
When Federated Learning Meets Blockchain: A New Distributed Learning Paradigm



Chuan Ma, Jun Li, and Long Shi
Nanjing University of Science and Technology, CHINA

Ming Ding
CSIRO, AUSTRALIA

Taotao Wang
Shenzhen University, CHINA

Zhu Han
University of Houston, USA, and Kyung Hee University, SOUTH KOREA

H. Vincent Poor
Princeton University, USA

Digital Object Identifier 10.1109/MCI.2022.3180932
Date of current version: 19 July 2022

Abstract—Motivated by the increasingly powerful computing capabilities of end-user equipment, and by the growing privacy concerns over sharing sensitive raw data, a distributed machine learning paradigm known as federated learning (FL) has emerged. By training models locally at each client and aggregating learning models at a central server, FL has the capability to avoid sharing data directly, thereby reducing privacy leakage. However, the conventional FL framework relies heavily on a single central server, and it may fail if such a server behaves maliciously. To address this single point of failure, in this work, a blockchain-assisted decentralized FL framework is investigated, which can prevent malicious clients from poisoning the learning process, and thus provides a self-motivated and reliable learning environment for clients. In

Corresponding author: Jun Li (e-mail: jun.li@njust.edu.cn).

this framework, the model aggregation process is fully decentralized and the tasks of training for FL and mining for blockchain are integrated into each participant. Privacy and resource-allocation issues are further investigated in the proposed framework, and a critical and unique issue inherent in the proposed framework is disclosed. In particular, a lazy client can simply duplicate models shared by other clients to reap benefits without contributing its resources to FL. To address these issues, analytical and experimental results are provided to shed light on possible solutions, i.e., adding noise to achieve local differential privacy and using pseudo-noise (PN) sequences as watermarks to detect lazy clients.

Future wireless networks are expected to require very low latencies and high reliability. Migrating machine learning (ML) to end-user equipments (UEs) promotes these requirements, giving them the capability of making decisions based on locally acquired data, even if it loses connectivity to the network. Since data available at a given end-user device is typically limited, the training of on-device ML models can benefit from data exchange among UEs [1].

However, directly exchanging data among UEs presents risks of privacy leakage and information hijacking [2]. To reduce this risk, federated learning (FL) has been proposed, which is an ML framework that trains an artificial intelligence (AI) model across multiple UEs holding local datasets. In particular, distributed UEs train ML models locally, sharing their model parameters with a central server where these local models are aggregated into a global model. In this way, FL allows UEs to cooperatively learn a global model without exchanging their data directly. FL has been applied in practical settings, including health care and autonomous driving [3].

Although FL offers advantages in latency and privacy enhancement, it suffers from several limitations. First, in the FL process, it is assumed that the aggregator is trustworthy and will make fair decisions in terms of user selection and aggregation. However, this assumption is not always satisfied in practical situations where a biased aggregator can intentionally favor a few selected UEs, thereby biasing learning performance [1]. Second, although the aggregator has access only to the models trained by its UEs, private client data can still be inferred from those models. Thus, if the aggregator is compromised, privacy leakage happens. Lastly, the conventional FL architecture is vulnerable to malicious clients that can attack learning via model poisoning [4].

As a secure technology, blockchain has the capability to tolerate a single point of failure with distributed consensus, and it can further implement incentive mechanisms to encourage participants to effectively contribute to the system [5]. For these reasons, blockchain has been introduced into FL to mitigate its aforementioned limitations. For example, [5] introduced a blockchained FL architecture to verify uploaded model parameters and investigated related system performance indices, such as learning delay and block generation rate. Moreover, [6] proposed a privacy-aware architecture that uses blockchain to enhance security when parameters of ML models are shared among UEs. In addition, the authors of [7] proposed a high-level framework by enabling encryption during model transmission, and [8] further applied this framework in a military setting. With the advanced features of blockchain, such as tamper-resistance, anonymity, and traceability, an immutable audit trail of ML models can be created for greater trustworthiness in tracking and proving provenance [9]. In addition, security and privacy issues arising in the decentralized FL framework are investigated in [6], [10], [11], which delegated the responsibility of storing ML models to a trust community in the blockchain. However, the assumption of a trust community may incur the same privacy issues when ML models are transmitted over the air, and the credibility of this community also needs further verification. In addition, these works either have not completely clarified and fully addressed incidental issues, such as the long learning delay and impact of blockchain forking on FL, or are difficult to apply.

In the present work, a blockchain-assisted decentralized FL (BLADE-FL) framework, which can overcome the single point of failure problem, is proposed in detail. In addition, several residual issues that exist in the BLADE-FL framework are further investigated, and related solutions are provided. The rest of this paper is organized as follows. The design of the BLADE-FL framework is presented in Sec. II, and residual issues, including privacy, resource allocation, and lazy clients, are investigated in Sec. III. In Sec. IV, extensive experimental results are provided to show the effectiveness of the corresponding solutions. Finally, promising future directions are suggested and conclusions are drawn in Sec.V.

II. BLADE-FL Framework

With the aid of blockchain, the aim is to build up a secure and reliable FL framework. To ensure this, the model updating process of FL is decentralized at each participating client, which is robust against the malfunction of traditional aggregators. In this section, the BLADE-FL framework, as well as how it achieves dynamic client selection and a decentralized learning aggregation process is presented.

The BLADE-FL framework is composed of three layers. In the network layer, the network features a decentralized peer-to-peer (P2P) network that consists of task publishers and training clients, wherein a learning mission is first published by a task publisher and then completed by the cooperation of several training clients. Different from previous work, in which model aggregation occurs in a trust community in the blockchain [5]–[11], a fully decentralized framework is realized in which each client must train ML models and mine blocks for publishing aggregating results. In the blockchain layer, each FL-related event, such as publishing a task, broadcasting learning models, and aggregating learning results, is tracked by blockchain. In the application layer, the smart contract (SC) and FL are utilized to execute the FL-related events. Next, the workflow and key components of the BLADE-FL framework are presented.

A. Workflow

As shown in Fig. 1, the workflow of the proposed framework consists of the following steps.

- ❑ **Step 1**: Task publishing and node selection. A task publisher broadcasts an FL task by deploying an SC over the blockchain network. In the deployed SC, the task publisher must deposit a reward as a financial incentive for the learning task. The SC selects available training nodes to participate in this learning task.
- ❑ **Step 2**: Local model broadcast. Each training client runs its local training by using its own data samples, and it broadcasts local updates and the corresponding processing information (e.g., computation time and local data size) over the P2P network. Privacy leakage may happen during this transmission, and this issue is further investigated in Sec. III-A.
- ❑ **Step 3**: Model aggregation. Upon receiving the local updates from other training nodes before a pre-set timestamp, each client updates the global model according to the aggregating rule defined in the SC.
- ❑ **Step 4**: Block generation. Each training client changes roles from trainer to miner and begins mining until it either finds the required nonce or receives a generated block from other miners. The learning results are stored in the block as well. When one miner generates a new block, other clients verify the contents of this block (e.g., the nonce, state changed by the SC, transactions, and aggregated model). The resource-allocation issue happens in each client in this step, and related discussions will be given in Sec. III-B.
- ❑ **Step 5**: Block propagation. If a block is verified by the majority of clients, this block will be added on the blockchain and accepted by the entire network. The lazy client issue occurs in this step and is further investigated in Sec. III-C.
- ❑ **Step 6**: Global model download and update. Each training client downloads the aggregated model from the block and performs updates before the next round of learning.
- ❑ **Step 7**: Reward allocation. The SC deployed by the task publisher rewards the training clients according to their contributions to the learning task.

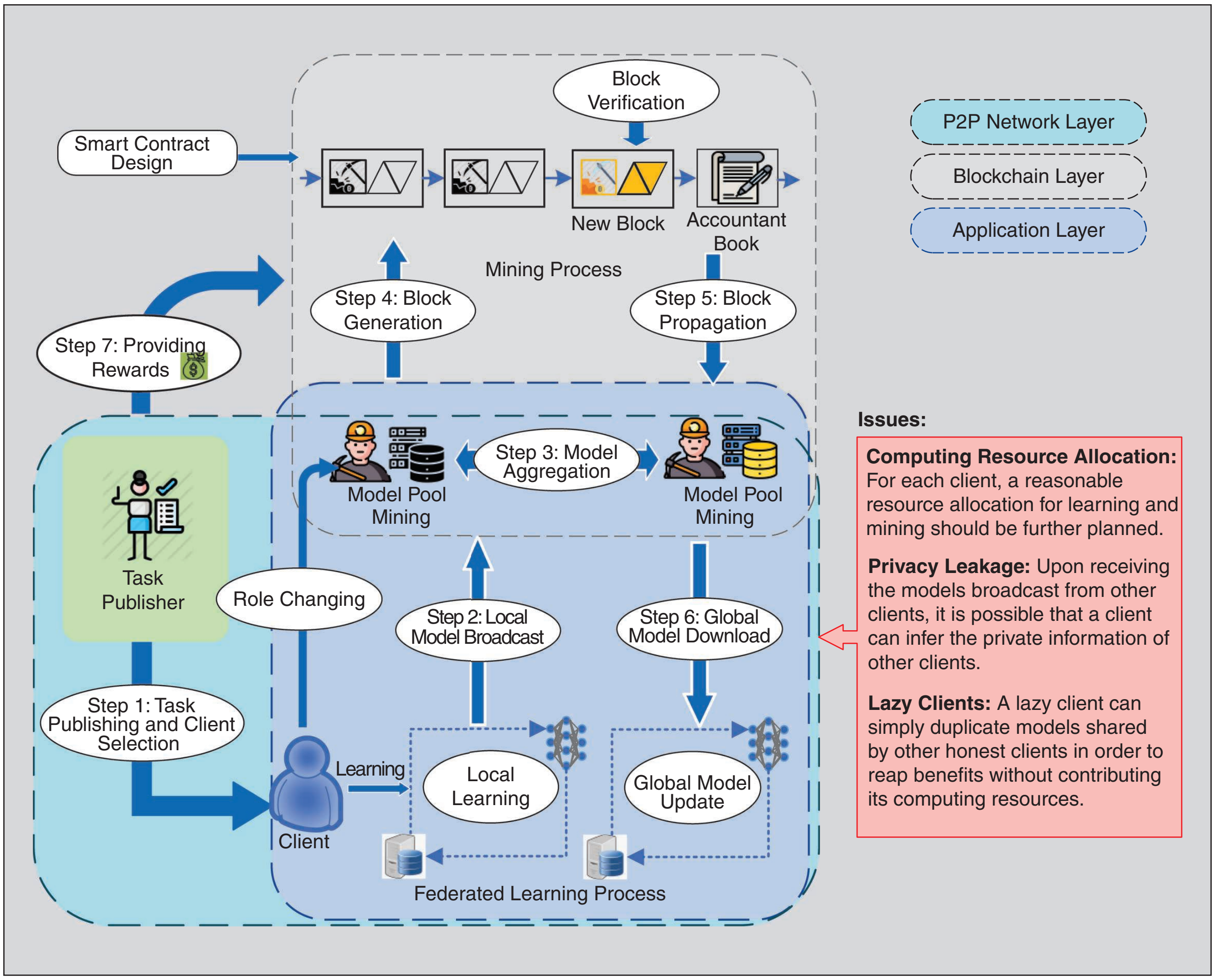

Block Verification
Smart Contract Design
New Block
Accountant Book
Mining Process
Step 4: Block Generation
Step 5: Block Propagation
Step 7: Providing Rewards
P2P Network Layer
Blockchain Layer
Application Layer
Model Pool Mining
Step 3: Model Aggregation
Model Pool Mining
Task Publisher
Role Changing
Step 2: Local Model Broadcast
Step 6: Global Model Download
Step 1: Task Publishing and Client Selection
Learning
Local Learning
Global Model Update
Client
Federated Learning Process
Issues:
Computing Resource Allocation: For each client, a reasonable resource allocation for learning and mining should be further planned.
Privacy Leakage: Upon receiving the models broadcast from other clients, it is possible that a client can infer the private information of other clients.
Lazy Clients: A lazy client can simply duplicate models shared by other honest clients in order to reap benefits without contributing its computing resources.

FIGURE 1 BLADE-FL workflow.

Before delving into each step, the key designs in BLADE-FL are elaborated as follows.

As a secure technology, blockchain has the capability to tolerate a single point of failure with distributed consensus, and it can further implement incentive mechanisms to encourage participants to effectively contribute to the system [5].

B. SC Design

SCs are self-executing contracts defining rules for negotiating, verifying the fulfillment of rules, and executing the agreement using formal code. The BLADE-FL framework relies on SCs to enable trusted dynamic client selections in terms of desired distributed learning services, without relying on a centralized authority. Moreover, BC-FL enables all clients to verify the learning results that are recorded on the blockchain, whereby distributed clients can be incentivized to participate and untrusted learning models can be detected. Based on the verification results, the reputation of each distributed client can be automatically updated, making the selection of learning nodes more reliable. In addition, the design of SCs in the BC-FL also includes the aggregating rules, and thus provides a fair and open rewarding of feedback for participating clients. The SC in BC-FL enables three main functions as follows.

Function 1: Learning task publishing. A task publisher broadcasts an FL task through an SC to all users. The SC contains the task requirements (e.g., the data size, training accuracy, latency, etc.), the aggregating rules, and rewards paid by the task publisher.

Function 2: Dynamic bidding for requests and automatic incentive. Distributed training nodes, acting as auctioneers, bid for the task by replying with their costs and capabilities. Note that to enforce accountability, each training client must stake a deposit to the SC. The task replies from training nodes are recorded on the blockchain by the SC. Then, the SC selects training clients with more valuable replies (e.g., higher capability and lower cost) as the bid winners to jointly execute the FL task. The training clients that lose the bidding will reclaim their deposits from the SC, while the deposits made by winners will be automatically refunded if the learning results are verified to be trustworthy afterwards.

Function 3: Learning results aggregation and rewards feedback. Before generating a new block, each client will aggregate the uploaded models according to the aggregating rule in the SC, in which the contribution of each one in the aggregated model is also recorded in the newly generated block. Then, the SC is automatically triggered to reward the miner that helps aggregate the learning model and the training clients that contribute to the FL process.

C. BLADE-FL Design

The main purpose of BLADE-FL is to enable trusted cooperative ML among distributed nodes. The decentralized accountability enables all miners to verify the quality of uploaded models that are recorded on the blockchain. In addition, distributed training nodes can be motivated to participate in the FL process and misbehaving ones can be recognized from the low-quality FL services they provide. The key steps follow.

Local model updating and uploading: Training nodes are bid winners with capable devices and available sets of data samples. In each learning iteration, each training node updates a local ML model in a parallel manner by using the global model and its local data samples, and broadcasts its local model in the network. In the present work, local updates can be received by all of the miners through the gossip protocol [12] over the P2P network. In this context, the aggregation process in traditional FL is decentralized to each client that stores the uploaded models in its respective model pool.

Model aggregation: After collecting the uploaded models in the pool, each client calculates the global model updates according to the aggregating rule in the SC. In the proposed architecture, the clients are designed to aggregate the learning parameters truthfully through a distributed ledger. Similar to the prevailing block structure in [6], each block in a ledger consists of body and header parts. Specifically, the body stores the local model updates, such as the local data size and computing time of the associated training node and the aggregated learning parameters. The header contains the information of a pointer to the previous block, block generation rate, and output value, such as the proof of work (PoW), in the consensus protocol.

Model recording and publishing: The clients record the aggregated models in their blocks and publish the recorded models by broadcasting the generated block to the entire network. The blocks can be generated by using distributed or lightweight consensus protocols, such as PoW, proof of stake (PoS), delegated PoS (DPoS), etc. [13]. In this paper, PoW is considered due to its strong security over decentralized networks, and a synchronous schedule is used to ensure that all of the miners start mining at the same time.

Once a client finds the hash value, its candidate block becomes a new block, and the generation rate of this block is controlled by the PoW difficulty. Then, this generated block is broadcast to all of the other miners in the framework. All of the other miners must verify the nonce and the aggregated results contained in this block. For example, clients can compare the aggregated results with the one in the publishing block or use a public testing dataset to justify the effectiveness of the uploaded models. If the verification result is correct, other clients will accept it as a legal block and record it; otherwise, others will discard this generated block and continue to mine the previous legal block.

Reward allocation: The task publisher provides learning rewards for the participating training nodes, and the

The decentralized accountability enables all miners to verify the quality of up-loaded models that are recorded on the blockchain.

volume can be proportional to the size of the training data. It is noted that the reward mechanism can be further amended by combining consideration of the data size and the quality of data samples. In this case, clients are responsible for verifying the trustworthiness of local updates after aggregation to address the situation that untruthful UEs may exaggerate their sample sizes with abnormal local model updates. Specifically, when clients calculate the rewards for each training node, they can give scores/reputations to the training nodes based on the model qualities. In the next aggregation, nodes with low scores will be given less weight, and they will be identified and gradually ignored during learning. In practice, this can be guaranteed by Intel's software guard extensions, allowing applications to be operated within a protected environment, which has already been used in blockchain technologies [14]. In addition, miners can also obtain rewards from mining and aggregating models, which can be treated as a gas tax in the traditional blockchain.

A task publisher first broadcasts an FL task through an SC to all of the clients. Consider that N clients dynamically bid for this task and use PoW as the consensus mechanism in the verification process. Thus, the pseudo-code of the proposed BLADE-FL framework is outlined in Algorithm 1.

ALGORITHM 1 BLADE-FL algorithm.

```
Data: Number of communication rounds $T$, initial model
      $\mathbf{w}^0$, and proximal term $\mu$ in local learning.
1  Task publishing and client selection.
2  Initialization: $t = 1$ and $\mathbf{w}_i^0 = \mathbf{w}^0$, $\forall i$
3  while $t \le T$ do
4      Local training:
5      while $i \in \{1,2,\dots,N\}$ do
6          Update local model $\mathbf{w}_i^{(t)}$ as
7              $\mathbf{w}_i^t = \arg\min_{\mathbf{w}_i}\left(F_i(\mathbf{w}_i) + \frac{\mu}{2}\|\mathbf{w}_i - \mathbf{w}^{t-1}\|^2\right)$.
8      Model broadcasting and aggregating:
9      Update aggregated model $\mathbf{w}^{(t)}$ as
10         $\mathbf{w}^t = \sum_{i=1}^{N} p_i \widetilde{\mathbf{w}}_i^t$.
11     Block mining and verification:
12     Each client starts mining its block that includes the
       aggregated model and verifies the generated block.
13     Global model downloading:
14     Each client downloads the aggregated model from the
       verified block and updates.
15     Reward allocation for learning and mining.
16     $t \leftarrow t + 1$
Result: $\widetilde{\mathbf{w}}^{(T)}$
```

III. Unique Issues and Potential Solutions

In this section, three critical issues that the proposed framework may be confronted with, namely, privacy, resource allocation, and lazy clients, are described.

A. Privacy

In BLADE-FL, the roles of each client include mining and training. To aggregate the global model, the trained local model will be published among clients, which raises privacy issues. Previous works [5]–[11] usually artificially assign the training and mining tasks to two disjoint sets of clients, and they widely adopt that the miners are always trustful. However, if an eavesdropper exists in the wireless environment, the published information of local models can cause privacy leakage. To address this, a differentially private mechanism can be implemented at the client side. In detail, the key steps are as follows.

- Each client sets up a self-required privacy level for itself before training. For example, the ith client may have a local privacy budget ϵ_i. Note that a small value of ϵ_i represents a high local privacy level, and it will induce more additive noises on the parameters.
- To achieve local differential privacy (LDP), each client will add a random noise that follows a certain distribution on the uploaded models. For example, a random Gaussian noise $N(0,\sigma^2)$ or a Laplace noise $Lap(\lambda)$ will be added. Note that a large noise power implies a high privacy level.
- Upon receiving the perturbed models, all of the clients can aggregate the global model locally and store it in the generated block. Because of the injected noise, the learning convergence as well as the system performance will be negatively affected. A trade-off between the privacy requirement and learning performance needs further investigation. In addition, a non-uniform allocation of additive noise over communication rounds may improve learning performance; for example, a decay rate for the noise power can be applied when the learning accuracy between two adjacent communication rounds stops improving [15].

B. Computing Resource Allocation

Since the computation resource is limited at each client, each participant must appropriately allocate the resources for local training and mining to complete the task. Specifically, more computing resources can be devoted either to accelerate model updating or block generation. To meet the specific task requirements, such as learning difficulty, accuracy, and delay, each node optimizes its allocation strategy to maximize its reward under constraints of local capability.

According to the constraints, the computing resource allocation can be formulated as an optimization problem under the accurate mathematical model, as follows, in detail.

- ❑ The block generation rate is determined by the computational complexity of the hash function and the total computing power of the blockchain network (i.e., total CPU cycles). The average CPU cycles required to generate a block can be defined as kc_B, where k denotes the mining difficulty and c_B is the average number of total CPU cycles required to generate a block. Thus, the average generation time of a block (t_B) can be expressed as kc_B/Nf, where N is the number of clients and f denotes the CPU cycles per second of each client.
- ❑ The training time consumed by each training iteration, t_T, can be expressed as $(|D|c_T/f)$, where $|D|$ denotes the number of samples of each client and c_T is the number of CPU cycles required to train one sample.
- ❑ Considering that a typical FL learning task is required to be accomplished within a fixed duration of T_{Sum}, the total learning and mining times should be satisfied that $K(\tau t_T + t_B) \leq T_{\text{Sum}}$, where K denotes the number of total communication rounds and τ the number of local training epochs. Thus, to achieve required learning performance, the number of communication rounds K should be optimized under a certain ratio between the training and mining time.

To achieve local differential privacy, each client will add a random noise that follows a certain distribution on the uploaded models.

C. Lazy Clients

As the verification is processed locally, a lazy client may not perform local learning and directly copy uploaded parameters from other clients to save its computing resources. As a result, the client can devote more mining resources to reaping more mining rewards with a higher probability. However, this action degrades learning performance. To investigate the effect of lazy nodes on the system performance, related experimental results are provided in Sec.V-D.

To address the lazy client issue, a signature process can be implemented at each client, which is based on the pseudo-noise (PN) sequence. The signature in BLADE-FL is resilient to noise perturbation because the lazy clients are likely to perturb the plagiarized local models to hide their misbehavior. This process can improve detection accuracy of lazy clients at the cost of negligible overhead to the system. The details of the process are the following.

- ❑ Before broadcasting the local updates, each client will produce a PN sequence of length L, where L is usually a very large number (larger than the number of model parameters), and a same length with model parameters is selected and added to the local updates. This PN sequence has a high self-correlation coefficient and is difficult to detect or re-produce by other clients. Minimally, the complexity of detecting the PN sequence should be much larger than that of training the neural network so as to deter the attempt to discover the used PN sequence.
- ❑ Upon receiving local updates from the other clients, each client will use its own PN sequence to check the correlation coefficient with the updates. If high peaks in terms of the cross-correlation coefficient exist, then the lazy clients will be detected.
- ❑ Once a lazy client is recognized by a local client, this client can publish the previously used PN sequence to others and invite other honest clients to verify this process. Then, any future updates from the lazy client might be discarded as punishments.

The pseudo-code of the PN sequence detection is shown in Algorithm 2.

IV. Experimental Results and Potential Solutions

In this section, several related experimental results are provided to show the issues in the multi-functional miner in the proposed BLADE-FL system.

A. System Setup

For each experiment, the original training data is divided into non-iid training sets, and locally computes a stochastic gradient descent (SGD) updated on each dataset, and then the server aggregates updates to train a globally shared classifier. The prototype is evaluated on the Fashion-MNIST and Cifar-10 datasets.

ALGORITHM 2 **PN Sequence Detection Algorithm.**

```
Data: Number of communication rounds $T$, number of clients $N$,
      detection threshold $\lambda$, and amplitude of PN sequence $\alpha$.
1  Initialize: $\overline{\mathbf{w}}^0$, $t = 1$
2  while $t <= T$ do
3      if $t = 1$ then
4          $i$-th client adds an additive PN sequence to the
           learned model as $\widehat{\mathbf{w}}_i^t = \mathbf{w}_i^t + \alpha \mathbf{S}_i^t$.
5          Calculates the cross-correlation value between model
           parameters and its own PN sequence as
           $\mathbf{C}_{i,j}^t = \mathbf{S}_i^t * \widehat{\mathbf{w}}_j^t = \mathbf{S}_i^t * (\mathbf{w}_j^t + \alpha \mathbf{S}_j^t)$.
6          Sums the cross-correlation value as $C_j^t = \sum_{i=1, i \neq j}^{N} C_{i,j}^t$.
7          Performs peak detection.
8          if $C_j^t \geq \lambda$ then
9              $\mathbf{w}_j^t$ is copied;
10         else
11             $\mathbf{w}_j^t$ is honest.
12     Discards models that are detected as lazy clients
13     Performs a global update as $\overline{\mathbf{w}}^t = \sum_{i=1}^{N^t} (|D_i| \widehat{\mathbf{w}}_i) / \left( \sum_{j=1}^{N^t} |D_j| \right)$,
       where $N^t$ is number of honest clients after discarding
       lazy clients.
14     $t = t + 1$
Result: $\overline{\mathbf{w}}^T$
```

In the following, the average results of 20 run experiments are collected. For the blockchain setup, the total computation resource is set to $T_{\text{Sum}} = 200$ for each training node, and the total number of clients to $N = 20$. In each communication round, each client uses t_B time resources to generate a block and t_T time resources to pursue a learning epoch, where $t_B = 2$ for all of the experiments. Letting $\theta = t_T/t_B$, a larger θ implies that the client allots more computing resources to learning in each communication round.

B. Investigation of Local Differential Privacy

Local differential privacy is applied to each client by adding random Gaussian noise to the uploaded models in each communication round. The testing accuracies of the Fashion-MNIST and Cifar-10 datasets are plotted in Fig. 2 with respect to different privacy levels ϵ. In addition, an adaptive noise decaying method is compared with the constant one, which will decrease the noise power when the accuracy stops increasing. The figure further shows that the system achieves higher performance with a larger value of ϵ, which is under weaker privacy protection, and the adaptive method can further improve the learning performance under the same level of privacy protection.

C. Investigation of Resource Allocation

In this subsection, the resource-allocation results and the training loss values with different ratios (θ) of both datasets are plotted in Fig. 3. The figure shows the system performance for different ratios as the number of total communication rounds increases. Usually, a lower loss function value represents better training performance. In detail, it can be found that there exists an optimal total communication round (K) for each computing ratio θ. For example, the smallest training-loss value can be obtained if clients stop learning in 14 communication rounds each with 15 learning epochs when $\tau = 1$ in the Fashion-MNIST dataset. Moreover, for different computing ratios, the optimal loss value tends to be different. This is due to the fact that the optimal number of local learning epochs varies according to different values of θ. In addition, similar trends can be found in the Cifar-10 dataset.

D. Investigation of Lazy Clients

In this subsection, the impact of lazy clients on the proposed framework is investigated. The signal-to-noise ratio (SNR) is used to denote the ratio of the power of original model parameters to

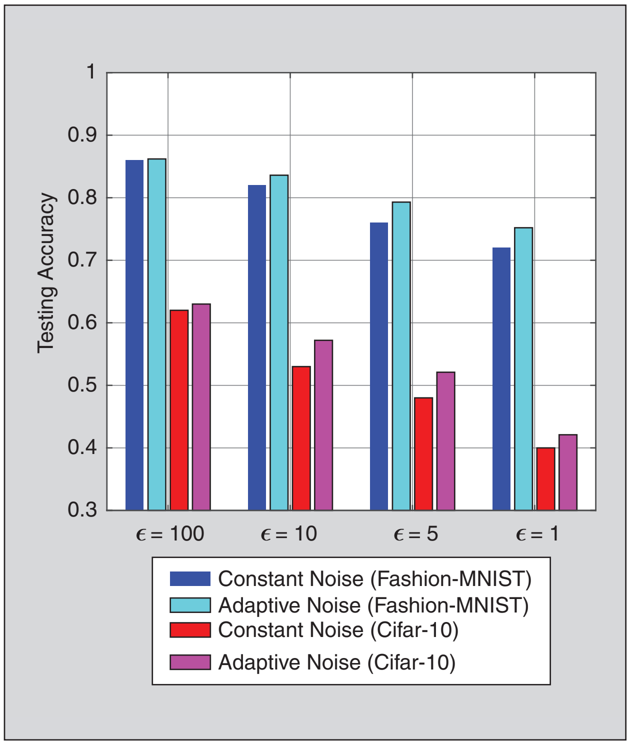


FIGURE 2 Learning performance with respect to different privacy levels.

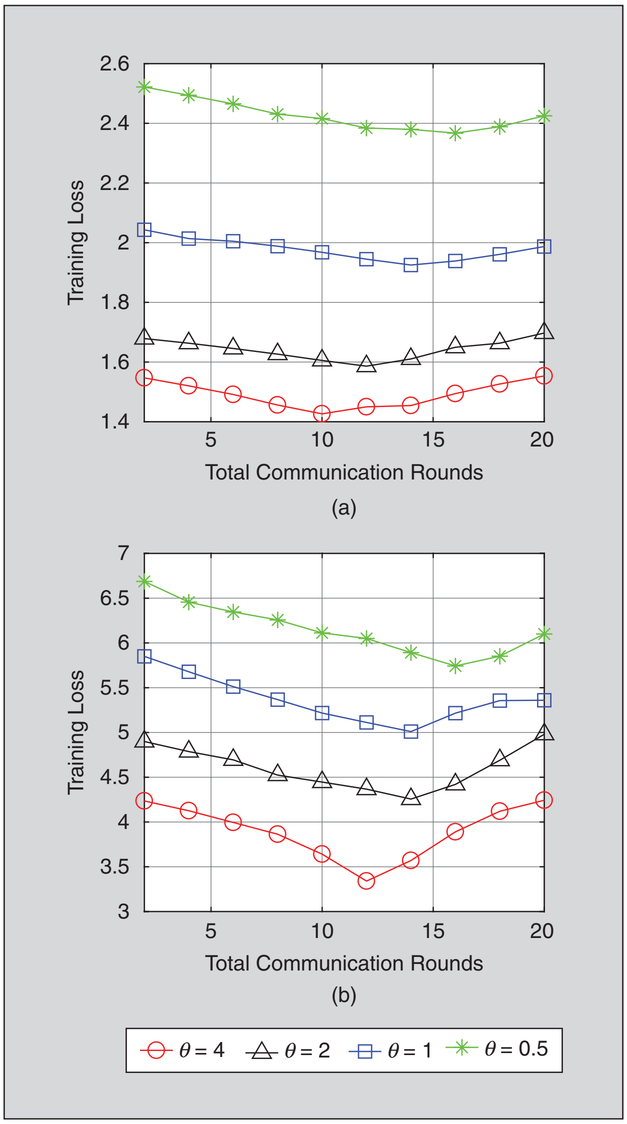


FIGURE 3 Learning performance of different total communication rounds under different resource-allocation ratios. (a) Fashion-Mnist (b) Cifar-10.

TABLE I **Detection rate with different PN sequence powers for Fashion-MNIST and Cifar-10 datasets.**

SNR	9 dB	6 dB	3 dB
Fashion-MNIST	0.931	0.989	0.999
Cifar-10	0.925	0.975	0.996

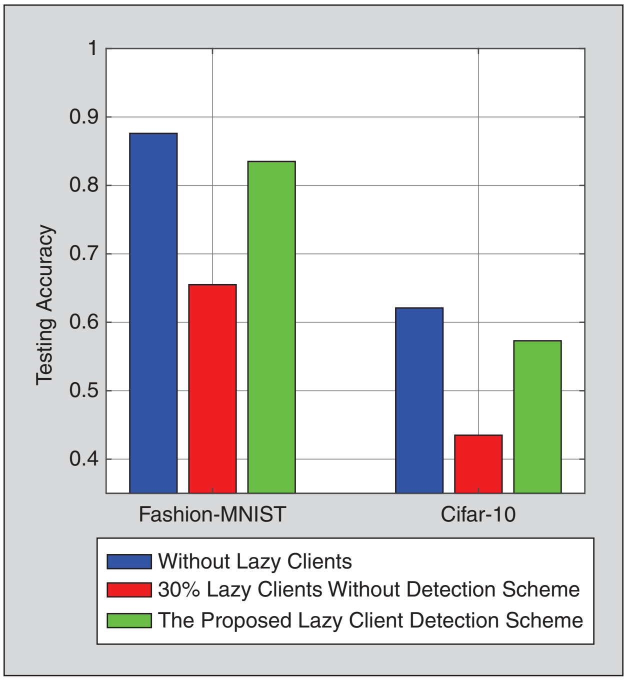


FIGURE 4 Learning performance with/without lazy client detection.

that of the injected PN sequence, and Table I represents the detection rate of lazy clients under different SNRs. If a high peaks in terms of the cross-correlation coefficient surpasses a predefined threshold, this client is identified as a lazy one. A PN sequence of length 2^{15} is generated and the first 25400 values are used to add onto the parameters. From the results with different SNRs, the detection performance is remarkable, and a nearly 100% rate of lazy client recognition when SNR=3 dB can be obtained. Fig. 4 shows the PN sequence-protection performance (SNR = 6 dB) when there are 30% (6) lazy clients in each communication round. As can be seen in this figure, the system performance with a certain percentage of lazy clients degrades sharply, i.e., 22.1% and 19.6% reduction for the Fashion-MNIST and Cifar-10 datasets, respectively. In addition, the proposed PN sequence-protection method achieves 18% and 13.8% performance gain for each dataset, respectively.

V. Future Directions and Conclusions

In this paper, the weaknesses of FL have been reviewed and a blockchain-assisted decentralized FL architecture, called BLADE-FL, has been proposed to address some of these weaknesses. The effectiveness of BLADE-FL has been shown in addressing these issues, notably the problem of a single point of failure that exists in conventional FL. In addition, further issues have been investigated for BLADE-FL including privacy, resource allocation, and lazy clients, and possible solutions have been provided to address those issues and explored with experiments. These results provide guidelines for the design of the BLADE-FL framework.

Some directions for further study in this area include asynchronous and heterogenetic investigations for different client capabilities, such as computing capability, training-data size, and transmitting diversity, and SC design, which provides reasonable reward allocation for training and mining. In addition, lightweight model transmission using quantization and sketch may be an alterative way of reducing the transmission cost.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant No. 62002170, and 61872184, and in part by the Fundamental Research Funds for the Central Universities under Grant No. 30919011274, in part by the Natural Science Foundation of Jiangsu Province under Grant BK20210331, in part by the Jiangsu Specially-Appointed Professor Program in 2021, in part by the Natural Science Fund of Guangdong Province under Grant 2020A1515010708 and the Natural Science Fund of Shenzhen under Grant JCYJ20210324094609027, and in part by the U.S. National Science Foundation under Grants ECCS-2039716, CNS-2107216 and CNS-2128368.

References

[1] C. Ma *et al.*, "On safeguarding privacy and security in the framework of federated learning," *IEEE Netw.*, vol. 34, no. 4, pp. 242–248, Jul./Aug. 2020, doi: 10.1109/MNET.001.1900506.
[2] Z. Liu, P. Longa, G. C. C. F. Pereira, O. Reparaz, and H. Seo, "Fourℚ on embedded devices with strong countermeasures against side-channel attacks," *IEEE Trans. Dependable Secure Comput.*, vol. 17, no. 3, pp. 536–549, May/Jun. 2020, doi: 10.1007/978-3-319-66787-4_32.
[3] Y. Chen, X. Qin, J. Wang, C. Yu, and W. Gao, "Fedhealth: A federated transfer learning framework for wearable healthcare," *IEEE Intell. Syst.*, vol. 35, no. 4, pp. 83–93, Jul./Aug. 2020, doi: 10.1109/MIS.2020.2988604.
[4] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Process. Mag.*, vol. 37, no. 3, pp. 50–60, May 2020, doi: 10.1109/MSP.2020.2975749.
[5] H. Kim, J. Park, M. Bennis, and S. Kim, "Blockchained on-device federated learning," *IEEE Commun. Lett.*, vol. 24, no. 6, pp. 1279–1283, Jun. 2020, doi: 10.1109/LCOMM.2019.2921755.
[6] Y. Lu, X. Huang, Y. Dai, S. Maharjan, and Y. Zhang, "Blockchain and federated learning for privacy-preserved data sharing in industrial IoT," *IEEE Trans. Ind. Informat.*, vol. 16, no. 6, pp. 4177–4186, Jun. 2020, doi: 10.1109/TII.2019.2942190.
[7] X. Bao, C. Su, Y. Xiong, W. Huang, and Y. Hu, "Flchain: A blockchain for auditable federated learning with trust and incentive," in *Proc. 2019 5th Int. Conf. Big Data Comput. Commun. (BIGCOM)*, pp. 151–159, doi: 10.1109/BIGCOM.2019.00030.
[8] P. K. Sharma, J. H. Park, and K. Cho, "Blockchain and federated learning-based distributed computing defence framework for sustainable society," *Sustain. Cities Soc.*, vol. 59, p. 102,220, 2020.
[9] S. Wang, "Blockfedml: Blockchained federated machine learning systems," in *Proc. 2019 Int. Conf. Intell. Comput., Autom. Syst. (ICICAS)*, pp. 751–756, doi: 10.1109/ICICAS48597.2019.00162.
[10] Y. Qu, S. R. Pokhrel, S. Garg, L. Gao, and Y. Xiang, "A blockchained federated learning framework for cognitive computing in industry 4.0 networks," *IEEE Trans. Ind. Informat.*, vol. 17, no. 4, pp. 2964–2973, Apr. 2021, doi: 10.1109/TII.2020.3007817.
[11] S. Otoum, I. Al Ridhawi, and H. Mouftah, "Blockchain-supported federated learning for trustworthy vehicular networks," Dec. 2020, pp. 1–6.
[12] M. Jelasity, "Gossip," in *Self-Organising Software*. New York, NY, USA: Springer-Verlag, 2011, pp. 139–162.
[13] L. Ismail, H. Materwala, and S. Zeadally, "Lightweight blockchain for healthcare," *IEEE Access*, vol. 7, pp. 149,935–149,951, 2019, doi: 10.1109/ACCESS.2019.2947613.
[14] P. Fairley, "Blockchain world - feeding the blockchain beast if bitcoin ever does go mainstream, the electricity needed to sustain it will be enormous," *IEEE Spectr.*, vol. 54, no. 10, pp. 36–59, Oct. 2017, doi: 10.1109/MSPEC.2017.8048837.
[15] K. Wei *et al.*, "User-level privacy-preserving federated learning: Analysis and performance optimization," *IEEE Trans. Mobile Comput.*, early access, Feb. 4, 2021, doi: 10.1109/TMC.2021.3056991.