Toward On-Device Federated Learning: A Direct Acyclic Graph-Based Blockchain Approach

Mingrui Cao, Long Zhang, and Bin Cao, Member, IEEE

Abstract—Due to the distributed characteristics of federated learning (FL), the vulnerability of the global model and the coordination of devices are the main obstacles. As a promising solution of decentralization, scalability, and security, leveraging the blockchain in FL has attracted much attention in recent years. However, the traditional consensus mechanisms designed for blockchain-like proof of work (PoW) would cause extreme resource consumption, which reduces the efficiency of FL greatly, especially when the participating devices are wireless and resource-limited. In order to address device asynchrony and anomaly detection in FL while avoiding the extra resource consumption caused by blockchain, this article introduces a framework for empowering FL using direct acyclic graph (DAG)-based blockchain systematically (DAG-FL). Accordingly, DAG-FL is first introduced from a three-layer architecture in detail, and then, two algorithms DAG-FL Controlling and DAG-FL Updating are designed running on different nodes to elaborate the operation of the DAG-FL consensus mechanism. After that, a Poisson process model is formulated to discuss that how to set deployment parameters to maintain DAG-FL stably in different FL tasks. The extensive simulations and experiments show that DAG-FL can achieve better performance in terms of training efficiency and model accuracy compared with the typical existing on-device FL systems as the benchmarks.

Index Terms—Anomaly detection, asynchrony, blockchain, direct acyclic graph (DAG), federated learning (FL).

I. INTRODUCTION

In order to solve the privacy problem caused by the data island and make the best use of distributed data on various devices, federated learning (FL) has recently drawn much attention, which is a distributed machine learning framework, and participants in FL transfer and communicate the model parameters without revealing user privacy to use their own data to establish machine learning models [1], [2]. For wireless scenarios, on-device FL is one of the most typical applications where participating nodes of FL are numerous mobile devices under a wireless network [3], [4]. Meanwhile, with the advent of the 5G era, mobile devices would have sufficient communication bandwidth [5], [6], which makes it possible to establish an efficient FL system on mobile devices.

Although FL is widely considered to be a feasible way to enhance privacy and security in 5G wireless networks, it still faces many challenges during deployment [7]. The main two points are given as follows.

1) Device Asynchrony: Various nodes have different resources for FL in terms of computing, communication, caching, battery power, data, and training time, which would result in heterogeneity and it is a natural characteristic especially in wireless networks [7]. As a result, due to the limited capacity and ability of node, network, and system, it is hard to coordinate the FL process perfectly generating the device asynchrony [8]. To this end, in the traditional centralized and synchronous FL system, such as Google FL proposed in 2017 [1], a single node must wait for other nodes to complete their tasks and then enter the next round together after completing its own training task. However, this manner might generate the deteriorated cost incurred by the bottleneck node obviously, in which the worst case is that, if a node shuts down during training, it may let a round of FL be invalid completely [9], [10].

2) Anomaly Detection: Due to privacy concerns, the local dataset and local operation process of a node are invisible to others, which makes FL suffer from abnormal actions of nodes easily. Especially considering massive participating nodes, the challenge of FL is to detect abnormal nodes and avoid adverse effects as much as possible. Abnormal nodes will reduce the overall efficiency of the FL system and model accuracy by uploading abnormal parameters during the FL process, the machine learning model built by all nodes together would be very vulnerable, and thus, anomaly detection in FL is necessary [11]–[13].

To this end, some studies focus on asynchronous FL framework to solve device asynchrony and anomaly detection strategy to mitigate the impact of abnormal nodes in the FL system. Recently, considering the asynchronous system and security protection, blockchain becomes a natural design that adopted in FL [14]; the reasons are twofold: 1) nodes can...
announce local model immediately without any asynchrony requirement and 2) blockchain miners can collect and validate model parameters to encourage normal action while avoiding anomaly.

Although recent blockchained FL works [3], [15], [16] have achieved some progress and advantages to address the mentioned challenges, there are still some problems that remained that have not been investigated thoroughly. First, although these works basically follow the synchronous framework of Google FL, they allow nodes not to wait for others. Thus, a pseudoasynchronous FL system is established, in which the innovation is to use miners to replace the central servers. However, due to the synchrony, the mobile device that acted as a miner should be associated with each other, which obeys the decentralization of blockchain while declining the performance of FL in terms of delay, convergence, accuracy, and so on. Second, to maintain the blockchain operation, most works use proof of work (PoW) [17], which allows the node to act as a miner consuming an amount of computing resource for consensus achievement. Meanwhile, in order to detect abnormal nodes, the miner should also verify the correctness of uploaded model parameters. As a result, the training/learning efficiency might not meet the expectation well caused by the blockchain cost since the overall resource is limited, especially for on-device FL under wireless networks.

Through the above observations, our concerned issue is whether blockchain is available to establish an efficient asynchronous on-device FL system without introducing too much extra resource consumption. Due to the evolution of consensus mechanism, we notice that directed acyclic graph (DAG) ledger technology promotes blockchain from synchronous to asynchronous bookkeeping using voting consensus mechanism without mining [18]. Inspired by these, we propose an asynchronous DAG empowered FL for purpose of efficiency and immunity, referred to as DAG-FL. The main contributions of this work are illustrated as follows.

1) To the best of our knowledge, DAG-FL is the first DAG-based FL framework forming an asynchronous updating to well solve problems of device asynchrony and abnormal nodes under wireless networks.

2) We formulate a theoretical model to analyze and discuss how to keep DAG-FL working stably and autonomously, which is helpful to understand the operation of DAG-FL.

3) We conduct extensive experiments in simulation computer systems and testbed deployment to evaluate the performance of DAG-FL compared with Google, asynchronous, and block FL systems to discuss the improvement and feasibility of the proposed DAG-FL.

The rest of this article is organized as follows. In Section II, we provide some basic concepts in DAG-FL. Section III proposes DAG-FL in detail. Furthermore, Section IV analyzes two details of deploying DAG-FL and discusses how to make DAG-FL work properly. Then, Section V analyzes the performance of DAG-FL in the case of large-scale nodes through simulation experiments and implements DAG-FL on a real testbed with a small number of nodes. As future work, we discuss the expansion of DAG-FL in Section VI.

In Section VII, we will briefly discuss some related works of existing FL systems from synchronous, pseudoasynchronous, and blockchained aspects. Finally, Section VIII gives a summary.

II. PRELIMINARIES

In order to elaborate on the proposed DAG-FL, the basic concepts of FL and blockchain involved in DAG-FL are introduced briefly in this section.

A. Federated Learning

FL is a distributed machine learning method to utilize data on different devices without privacy leaking, and it is proposed by Google in 2017 officially [2]. In order to achieve the model coconstruction, the basic principle of FL is to update the global model by collecting local models on different nodes. In terms of the global model updating methods, the FL framework can be divided into three categories, which are synchronous FL, pseudoasynchronous FL, and asynchronous FL.
Traditional FL, such as Google FL, is composed of a central server and numerous nodes, as shown in Fig. 1(a), where the global model is maintained on the central server and FL iterations are performed on nodes. In each round, the central server selects several idle nodes to assign FL tasks. The selected nodes then begin to run FL iterations. First, the node downloads the current global model from the central server. Second, the node trains the global model using local data to get a local model and upload it to the central server. Finally, when the central server collects all local models uploaded by selected nodes, it runs the FederatedAveraging algorithm [1] that combines all the local models on averaging to get an aggregated new global model. This global model updating method is a typical synchronous manner, and the traditional FL, such as Google FL, is so-called synchronous FL.

By uploading trained local models to the central server immediately, the pseudoasynchronous FL [19] is constructed, as shown in Fig. 1(b). Nodes in pseudosynchronous can download the global model from the central server and train it with local data whenever it is in the idle state. The central server will update the global model periodically by aggregating the newly uploaded local models in the cache. To accelerate the system efficiency, pseudoasynchronous FL should set a small interval between global model updating. However, data on bottleneck nodes cannot be fully utilized and results in a final target model with less prediction accuracy. In addition, taking the miner as the central server and the block as the cache, blockchained FL [3], [15] can be easily built with the pseudoasynchronous framework.

In fact, due to device asynchrony [7], neither synchronous nor pseudoasynchronous FL could fit the on-device FL scenarios well, and thus, asynchronous FL has been studied recently. In asynchronous FL, it only needs two steps to complete an iteration of FL. First, any node can download the global model from the central server whenever it is idle and train the global model to get a local model. Then, the node uploads the local model back to the central server, and the central server updates the global model immediately whenever the local model is collected [9], as shown in Fig. 1(c). Furthermore, asynchronous FL allows global models to be updated by local models trained from the latest and outdated global models.

B. Blockchain and DAG Ledger Technology

Blockchain is a peer-to-peer (P2P) distributed ledger technology for storing information securely and immutably [20]. Usually, the traditional blockchain is based on Proof of X (the most famous one is PoW [17]), which consumes much resource on solving hash problems to decide which block to add to the chain next while slowing down the access rate of new blocks to avoid fork problem [21]. Moreover, the RAFT-based blockchain does not tolerate the existence of malicious nodes, and PBFT-blockchain meets the challenge of communication overhead increasing with the participating nodes exponentially [22].

In order to break the above limitations, DAG-based blockchain (DAG ledger) is proposed to promote the synchronous blockchain to asynchronous bookkeeping. The principle of DAG-based blockchain is to attach the new transactions in a forking topology [18] without maintaining the main chain, and thus, any new arrival transaction can be recorded in blockchain immediately without any coordination [23]. The consensus used in DAG-based blockchain can be treated as a voting mechanism, which requires nodes to validate and approve some early published transactions before publishing their own. The interval between the publishing time of a transaction and the current time is called staleness. Transactions with good staleness, which are not approved yet on DAG, are usually called tips [24]. To publish a transaction on DAG, one node needs to go through three stages. In the first stage, the node selects some tips according to some algorithms or just randomly. In the second stage, the node validates the authentication and correctness of selected tips. In the final stage, a new transaction, composed of essential information and approvals to selected tips, is constructed and published on DAG. Through these three stages, the votes are stored in the published transactions, and unidirectional connections among transactions are built forming the DAG architecture through the approval relationship.

III. DAG-FL

In this section, we first provide the overview of DAG-FL and then introduce its operation process.

A. Asynchronous Architecture

DAG-FL is proposed as a decentralized asynchronous FL system, including application, DAG, and FL layers from top to bottom. To well elaborate, this hierarchical structure of DAG-FL is shown in Fig. 2.

1) FL Layer: The FL layer is the bottom layer to provide FL function. In order to obtain local models recorded as transactions on DAG, the FL layer allows any participating node to use its own data to train the global model. The global model is formed by using FederatedAveraging algorithm to aggregate local models stored in transactions on DAG. After training the global model, a new trained local model would be processed and published as a transaction on DAG.

2) DAG Layer: In the DAG layer, each node maintains a local DAG, where the transaction contains authentication information, local model parameters, and approval connections. The local DAG can be updated by broadcasting or inquiring through the wireless network, and thus, the new transaction (or say the new model) can be spread throughout the DAG-FL network finally.

3) Application Layer: The application layer is deployed on the top of DAG-FL, which provides the interface to external agents by running smart contracts. Through the smart contract, external agents can release FL tasks to nodes, observe the FL process, and obtain the target model as soon as FL is completed. When a specific FL task is released according to a smart contract, nodes in the DAG layer can participate in this work based on an incentive mechanism to gain a number of rewards [16]. Then, during the FL process, the smart contract will observe transactions on DAG to determine whether a target model has been published.
Accordingly, the application layer provides an interface for external agents to deploy DAG-FL easily, and the DAG and FL layer form an asynchronous platform for FL. Nodes with device asynchrony, such as smartphones and IoT devices in DAG-FL, all need to maintain a local DAG to record transactions published by every node. The local DAG on each node is updated by communicating with adjacent nodes periodically, and thus, newly published transactions can be seen by all nodes. As there is no central server in DAG-FL, one node in DAG-FL constructs a global model from its local DAG to iterate FL instead of requiring a global model from the central server. This feature promises that a node in DAG-FL can immediately participate in an iteration of FL whenever it is in the idle state. When the node completes an iteration of FL and gets a new trained local model, the new local model can be published on its local DAG as a transaction immediately, and later, the new published transaction would be seen by all other nodes. In this manner, the operation of any node cannot affect the state of other nodes, which would satisfy the asynchrony of mobile devices.

B. Consensus-Based Anomaly Detection

We propose a DAG-FL consensus to keep the asynchronous FL platform stable and present an effective way for anomaly detection in DAG-FL.

Traditional blockchained FLs often use miners to update the blockchain by running the consensus mechanism, such as PoW, enabling every node to observe transactions published on the blockchain. Similarly, as there are no miners in DAG-FL, each node in DAG-FL should both perform FL tasks and update the DAG by running the DAG-FL consensus. Based on the voting mechanism of DAG ledger technology, the DAG-FL consensus approves the nodes by validating both the authentication and local model correctness of tips. The authentication [25] of transactions can be validated by cryptography technology, such as RSA [26] in blockchain fields, and the local model can be simply validated by computing the accuracy with a test set formed by the local data. Whenever a node in DAG-FL performs one iteration of FL, it runs DAG-FL consensus and should first choose some tips on its local DAG to validate. Authenticated transactions with higher accuracy of local models would be chosen to construct the global model. The node then uses a local dataset to train the global model to get a local model. Finally, a new transaction that contains the newly trained local model is published and approves the tips that are used to construct the global model. Unlike the PoW consensus in other blockchained FL systems that consume resources on solving hash cryptography problems, DAG-FL consensus avoids the extra consumption of resources unrelated to FL.

As for security problems, nodes in DAG-FL can authenticate each transaction by using cryptography technology. Since authentication validation enables DAG-FL to avoid transaction impersonation, abnormal nodes doing Sybil attacks for network flooding could not impersonate other normal nodes to publish crafted transactions. In this way, any node is restricted to issue a large number of transactions in a short time to prevent Sybil attacks.

Although cryptography technology can preserve DAG-FL from the attack in blockchain, it has no resistance to malicious activities of abnormal nodes by publishing transactions with wrong local models in FL, where abnormal nodes include lazy nodes (publish transactions with stale local models) and malicious nodes (publish transactions with harmful local models). To detect these abnormal nodes, we alter the voting mechanism of DAG ledger technology and enable DAG-FL to evaluate the local model on each transaction.

DAG-FL consensus combines the voting mechanism of DAG ledger technology with the process of local model validation in FL, which can effectively detect abnormal nodes and mitigate their impact on FL. With the continuous extension of transactions on DAG, every approval of a transaction means that the local model on the approved transaction is selected to form a global model and influences the target model coconstruction of FL. Consequently, the more approvals a transaction gets, the greater impact it will have on FL; otherwise, it will be isolated and has less impact on FL. Due to this unique consensus, the machine learning model in DAG-FL is always trained toward the direction that most nodes expect, and we assume that most nodes in DAG-FL are normal nodes, while only a few nodes are abnormal. Abnormal transactions published by abnormal nodes usually have less prediction accuracy on the test set than that of transactions published by normal nodes. Thus, compared with normal ones, the probability of abnormal transactions being approved by subsequently published transactions is much smaller. During the process of FL, abnormal transactions are isolated, and their impact is minimized. In addition, nodes with too many isolated transactions can be detected by the DAG-FL as abnormal nodes, and then, DAG-FL can react to these abnormal nodes.

C. FL Algorithm

DAG-FL is an asynchronous FL system without any central server, so we design a special FL algorithm to perform FL iterations. This part will give some numerical definitions and introduce the FL algorithm of our DAG-FL.
Our DAG-FL is deployed on nodes that are mobile devices under a wireless network, such as smartphones, wearable devices, and IoT devices. We assume that these nodes can communicate with each other considering an average communication bandwidth $B$ under the wireless network. Let the set of mobile devices be denoted as $D = \{1, 2, 3, \ldots, N_D\}$ with $|D| = N_D$. $D_i$ is the $i$th node in $D$, and the set of training data on $D_i$ is denoted as $S_i$ with $|S_i| = N_i$, where $N_i$ is the number of samples in $S_i$. Considering different computing capacities of nodes, let $f_i$ be the processor frequency of $D_i$, used to represent the computing ability. $D_i$ should create a local DAG $g_i$ that is only visible to itself, and $g_i$ can be periodically updated. Let the model stored in the transaction be denoted as $\omega$. Thus, the local model trained by $D_i$ at time $t$ can be denoted as $\omega^t_i$.

In order to build a common machine learning model with DAG-FL, e.g., a two-layer CNN model, usually thousands of FL iterations are requested. Initially, $D_i$ starts an FL iteration at $t_0$ by validating some tips on its local DAG first and then chooses $k$ tips with local models $\omega^1_{d_1}, \omega^2_{d_2}, \ldots, \omega^k_{d_k}$ ($t_1, t_2, \ldots, t_k \leq t_0, d_1, d_2, \ldots, d_k \in D$) to aggregate a global model $\omega^t$ using the FederatedAveraging algorithm

$$\omega^t = \sum_{i=1}^{k} n_i \omega^t_{d_i}$$ (1)

where $\sum_{i=1}^{k} n_i = 1$, and $n_i$ is the weight factor representing the importance of local models; to simplify the FL algorithm of our DAG-FL, here, we set $n_i = 1/k$, which means that each local model is equally important.

After getting the global model, $D_i$ extracts $m$ samples from dataset $S_i$ as a minibatch $z_i$ to train the global model for $\beta$ epochs. Once $D_i$ gets a new local model $\omega^t_{d_i}$ trained by $\omega^t$, a transaction with $\omega^t_{d_i}$ is published on $g_i$, and the FL iteration is completed. Samples in minibatch $z_i$ can be denoted as $(x_i, y_i)$, where $x_i$ is the feature set and $y_i$ is the label set. Then, the loss function in machine learning can be denoted as $f_i(z, \omega)$, where $f_i(z, \omega) = l(x_i, y_i; \omega)$, and it is the prediction error of $(x_i, y_i)$ in $\omega$.

In addition, due to the incentive mechanism of DAG-FL, $D_i$ in DAG-FL expects to train global models in every participated FL iteration to get the local model $\omega$, which can minimize $F(\omega)$ as follows:

$$\min F(\omega) = E_{z \sim S_i} f_i(z, \omega).$$ (2)

For the whole DAG-FL system, external agents expect to get target model $\omega$ through smart contract after thousands of FL iterations to minimize $F(\omega)$ as follows:

$$\min F(\omega) = \frac{1}{N_D} \sum_{i \in D} F_i(\omega_i).$$ (3)

D. DAG-FL Operation

After a brief overview of DAG-FL, two algorithms involved in DAG-FL will be proposed in this part to introduce the operation process of DAG-FL in detail. For clarity, we elaborate the operation work flow of DAG-FL in Fig. 3.

The task publisher of FL in DAG-FL can be denoted as an external agent $E$, which can be regarded as an authoritative organization in the application layer holding a virtual machine to run the smart contract. Through the smart contract, $E$ executes DAG-FL controlling algorithm, including three main functions, which are: 1) initializing the machine learning model to publish an initial transaction; 2) periodically updating a local DAG to observe the FL process; and 3) and informing every node to terminate FL task after the target model is obtained. In detail, when $E$ releases an FL task specifying the structure of the machine learning model with an expected final prediction accuracy $ACC_0$, nodes in $D$ are applied to participate in this FL task. At the beginning time of the FL task $t_0$, $E$ publishes the initial transaction with $\omega^0_{d_0}$. After $t_0$, $E$ updates its local DAG $g_0$ and randomly selects $a$ tips within $T_{\max}$ to validate. After validation, $k$ tips with the highest accuracy will be selected to construct a global
model \(\omega_0\). If prediction accuracy of the \(\omega_0, \text{ACC}_t \geq \text{ACC}_0\), 
\(E\) broadcasts the end signal to all participating nodes in 
DAG-FL to terminate the FL, and \(\omega_0\) is the final target model. 
Otherwise, \(E\) will repeat to update \(g_0\) until the proper \(\omega_0\) is 
found.

**Algorithm 1** DAG-FL Controlling. \(\text{ACC}_0\) Is the Expected 
Prediction Accuracy of Target Model, \(D\) Is the Set of Nodes, 
\(g_0\) Is the Local DAG on \(E\), and \(\tau_{\text{max}}\) Is the Threshold Staleness 
of Tips

**External agent** \(E\) executes:

1. **Input** \(\text{ACC}_0\)
2. \(\omega_0^0 \leftarrow \text{Initialize the machine learning model}\)
3. Publish initial transaction including \(\omega_0^0\) to \(D\)
4. **while** true **do**
5. Update \(g_0\)
6. Validate \(\alpha\) tips on local DAG \(g_0\) within \(\tau_{\text{max}}\)
7. \(\omega_0 \leftarrow \text{Select \(\alpha\) tips with the highest accuracy to compute}\)
   global model according to (1)
8. \(\text{ACC}_t \leftarrow \text{Get accuracy by } \omega_0\)
9. **if** \(\text{ACC}_t > \text{ACC}_0\) **then**
10. Send end signal to \(D\)
11. **break**
12. **end if**
13. **end while**
14. **Output** \(\omega_0\)

**Algorithm 2** DAG-FL Updating. \(D\) Is the Set of Nodes, \(g_i\)
Is the Local DAG on \(D_i\), \(\tau_{\text{max}}\) Is the Threshold Staleness 
of Tips, and \(S_i\) Is the Local Dataset of \(D_i\)

**Node** \(D_i\) \((D_i \in D)\) executes:

1. **while** true **do**
2. **if** end signal is received **then**
3. **break**
4. **else**
5. Update \(g_i\)
6. **if** idle state **then**
7. Validate \(\alpha\) tips on local DAG \(g_i\) within \(\tau_{\text{max}}\)
8. \(\omega_i \leftarrow \text{Choose \(\alpha\) tips with the highest accuracy to compute}\)
   global model according to (1)
9. \(\omega_i^j \leftarrow \text{Train } \omega_i\) with \(S_i\) for \(\beta\) epochs
10. Publish the new transaction including \(\omega_i^j\) and 
   approvals on \(g_i\)
11. **end if**
12. **end if**
13. **end while**

After \(t_0\), all the participating nodes run the **DAG-FL updating** 
algorithm to achieve DAG-FL consensus whenever they 
are in the idle state. As nodes in DAG-FL are mobile devices, 
nodes may shut down during one iteration of FL because of 
device asynchrony. Any node in DAG-FL that is in the idle 
state should go through four stages to participate within one 
iteration:

1) **Stage 1**: The node selects some tips (no more than \(\alpha\))
within an appropriate staleness \(\tau_{\text{max}}\) from local DAG randomly.

2) **Stage 2**: The node first validates the authentication of tips 
selected in stage 1. Then, the node computes the prediction 
accuracy of the local models in the selected tips using its own 
test dataset.

3) **Stage 3**: The node chooses \(k\) \((k < \alpha)\) tips selected in the 
first stage with the highest accuracy to run **FederatedAveraging** 
algorithm and gets a global model. The node utilizes a local 
dataset to train the global model and gets a trained local model.

4) **Stage 4**: A new transaction is constructed. The new trans-
action contains authentication information, the local model 
trained in the third stage, and the approval information to 
approve the \(k\) tips chosen in the third stage.

After the completion of the above four stages, the node then 
successfully finishes one iteration in DAG-FL and publishes 
the new transaction to the DAG, which will be soon observed 
by all other nodes in DAG-FL.

Suppose that \(D_i\) is in the idle state at \(t_1\) and intends to 
perform an iteration of DAG-FL, as shown in Fig. 4; we 
take node \(D_i\) in DAG-FL as an example to illustrate the tips’ 
selection and transactions extension on \(g_i\) when running the 
DAG-FL updating as follows.

1) As shown in Fig. 4, \(D_i\) should select some recently 
published and has not been yet approved transactions 
(a.k.a tips) from its local DAG \(g_i\) to validate them.

2) Using tips’ selection, \(D_i\) first computes the staleness 
of all unapproved transactions seen on \(g_i\).

3) After computing the staleness, isolated transactions and 
staile tips with staleness over the threshold \(\tau_{\text{max}}\) are 
filtered out and a set of candidate tips remains.

4) Then, \(D_i\) randomly chooses \(\alpha\) tips from the set 
of candidate tips to validate.

Although this random selection strategy is simple, it would 
convinve lazy and malicious nodes to issue abnormal transac-
tions though they have been detected effectively. Nevertheless, 
we still adopt this tips’ selection strategy in this article for 
easy understanding and implementation. Moreover, we have 
provided some discussions on more advanced tips’ selection 
strategies in Section VI-B.

Following the tips’ selection, \(D_i\) validates the authentication 
of these \(\alpha\) selected tips and computes the prediction accuracy 
of models on them using part of its local dataset. Then, \(k\) tips 
with authentication and the highest prediction accuracy among 
the \(\alpha\) tips are utilized by \(D_i\) to computes the global model \(\omega_k\) 
using the **FederatedAveraging** algorithm.

After \(D_i\) trains \(\omega_k\) with local dataset \(S_i\), for \(\beta\) epochs, 
a new local model \(\omega_i^j\) is built, at time \(t_2\). Finally, a new transaction 
contains \(\omega_i^j\) and approval information are published to \(g_i\).

As DAG-FL is a completely decentralized asynchronous 
FL system without any central server, a global model can 
only be temporarily constructed from the local DAG. Thus, 
DAG-FL allows nodes with different global models to train 
at the same time during the FL process. Global models differ 
a lot from each other in the early stage of FL, which causes 
a low convergence rate of the target model. However, these 
temporarily constructed global models tend to be almost the 
same in the later stage of FL, which promises an impressive 
convergence rate just as classic asynchronous FL.
In summary, the delay of the whole iteration $h$ can be approximately expressed as

$$h = d_0 + d_1.$$  \hspace{1cm} (7)

Substituting (7) into (4), we can get $L_0$ as follows:

$$L_0 = \frac{k \lambda (\eta_0 \times \varphi_0 \times s \times \beta + \eta_1 \times \varphi_1 \times \alpha)}{(k - 1) f}.$$  \hspace{1cm} (8)

For a specific FL task, $h$ and $\lambda$ can be regarded as constant values. Thus, according to (8), $L_0$ is mainly determined by $k$ and $\alpha$. If tips in DAG-FL are too large, we can set a larger $k$ to reduce $L_0$ and keep DAG-FL running stably. It is worth noting that $k < \alpha$, and the setting of $k$ and $\alpha$ is also related to the immunity of abnormal nodes in DAG-FL. For fixed $\alpha$, we can observe that the larger the $k$ will lead to less probability that a transaction published by an abnormal node is isolated. Thus, when deploying DAG-FL to run a specific FL task, we can weigh and balance $k$ and $\alpha$ to keep DAG-FL operating with a proper $L_0$.

B. Staleness of Transactions

The interval between the publishing time of a transaction and the current time is called staleness [9], denoted as $r$. It is obvious that the longer interval causes the worse staleness. Generally, the transactions with better staleness can build a global model with better progress of FL.

With the assistant of the global model of FL, we can train a local model that is closer to the target model. If a node uses local dataset to train a global model constructed by tips with bad staleness and gets a local model, the new transaction that contains the trained local model is more likely to be isolated on DAG. These isolated transactions published by normal nodes are not conducive to the machine learning model convergence and significantly slow down the establishment of the target model of FL. In order to get a global model that can well represent the progress of FL, we need to select tips with good staleness on DAG to construct the global model in an iteration of DAG-FL.

In order to prevent nodes in DAG-FL from using transactions with bad staleness when constructing the global model, we set a threshold $\tau_{\text{max}}$ of staleness for the tips on DAG. Consequently, nodes will select tips with staleness no more than $\tau_{\text{max}}$ to construct the global model for training. Once the staleness of a transaction exceeds $\tau_{\text{max}}$, the transaction cannot be used as a tip on local DAG anymore. It should be noted that, though a small $\tau_{\text{max}}$ can ensure that tips all have good staleness, the number of tips $L_0$ may sharply reduce, which is also not conducive to the normal operation of DAG-FL. Thus, we need to weigh and balance the $\tau_{\text{max}}$ when deploying DAG-FL to solve a specific FL task.

V. SIMULATION AND IMPLEMENTATION

In order to evaluate the performance of DAG-FL, in this section, we first design a simulation platform to verify the performance of DAG-FL with large-scale participating nodes and then conduct the actual deployment of DAG-FL on a small number of nodes using the proposed algorithm.
A. Simulation

1) Platform Setting: We design a simulation platform pySimuFL that can evaluate the performance of the proposed DAG-FL compared to Google FL (with synchronous FL framework) [1], asynchronous FL (with asynchronous FL framework) [9], and block FL (with pseudosynchronous FL framework) [3]. For simulation, there are 100 nodes with device asynchrony in our pySimuFL that shares the same wireless network with a radius of 10 km. In order to illustrate the performance of DAG-FL more comprehensively, we consider a CNN task for image classification and an LSTM task for language modeling in the following simulation experiments.

For the CNN task, a CNN model proposed in [1], which has two 5 × 5 convolution layers (the first with 32 channels, the second with 64, and each followed with 2 × 2 max pooling), a fully connected layer with 512-unit ReLu activation and a final softmax output layer are conducted using the MNIST dataset. The MNIST dataset of handwritten digits has a training set of 60,000 samples and a test set of 10,000 samples. To reflect the non-IID feature of local data on mobile devices, we design a special scheme to distribute training data to every node. At first, we sort 2/3 training set by digit label, divide it into 200 shards of size 200, and assign each of 100 nodes 2 shards. Then, the remaining training set is equally assigned to 100 nodes. By using this data distributing scheme, each node is assigned a local dataset, including most samples of two exact digits and some samples of the other digits. In addition, the learning rate of this CNN model is set to be 0.002, and the loss function is cross entropy.

For the LSTM task, a stacked character-level LSTM language model [1] is conducted using the Shakespeare dataset to predict the next character of a speaking line. The LSTM model takes a series of characters with a length of 80 as input and embeds each of these into a learned 8-D space. The embedded characters are then processed through two LSTM layers, each of which has 256 nodes. Finally, a softmax output layer is connected to the second LSTM layer with one node per character. The Shakespeare dataset is split into a train set with 3,564,579 characters and a test set with 870,014 characters. As the Shakespeare dataset itself is composed of 1146 roles’ speakings, which is already highly unbalanced, the non-IID feature of local data is realized by randomly assigning the training set to 100 nodes. The LSTM model uses the cross-entropy loss function and is set with a learning rate of 0.3. To better compare the convergence difference among FLs in the following simulation experiments, the LSTM model has been pretrained to achieve an accuracy of 0.2518.

Considering a quasi-static network environment, we set up a relatively conservative bandwidth of wireless networks, i.e., 100 Mb/s. Nodes are set to be in the idle state for FL at different times, thus enabling one node on average ready for an FL iteration per second. Other parameter settings of the simulation platform are listed in Table I.

We consider normal and abnormal nodes in pySimuFL, where abnormal nodes can be set as lazy nodes and malicious nodes. Lazy nodes upload existing models instead of models trained with local data in an FL iteration, aiming to obtain potential rewards of FL. Malicious nodes include two kinds of nodes that are poisoning nodes and backdoor nodes that can launch either poisoning model attacks or backdoor attacks. Poisoning nodes are set to have wrong data for training, which can poison the global model in FL and aim at reducing the overall target model performance [13]. Backdoor nodes are those that launch targeted attacks (also called backdoor attacks) and aim to mislead the target model in FL. As the LSTM model using the Shakespeare dataset is vulnerable to targeted attacks, backdoor nodes are only concerned in the CNN task. For the CNN task in our pySimuFL, backdoor nodes modify part of their local MNIST pictures by replacing a 5 × 5 white square into the upper left corner [27] and want to mislead the final target model of FL to recognize pictures with a white square to the wrong digit label, which is the true digit plus one.

The results of Google FL, asynchronous FL, and block FL in the following figures are obtained by using the same nodes’ settings as DAG-FL. Note that here one iteration refers to the process that one node uses its local dataset to train the global model for β epochs. To compare the performance differences of the several FL systems fairly, some settings of Google FL, asynchronous FL, and block FL are presented as follows. For Google FL, in each round, ten nodes in the idle state are selected to download and train the global model from the central server, which means that Google FL will run ten iterations in a round. Asynchronous FL used in pySimuFL updates the global model by averaging the last global model with the newly uploaded local model. For the iteration process of block FL, we set up 100 nodes and five extra miners; 100 nodes in block FL are divided into five groups, and each group is associated with a miner. Every miner has the whole test dataset (100,000 picture samples for the CNN task and 10,740 80-length character line samples for the LSTM task) to validate published transactions. Whenever a miner collects five transactions published by nodes or waits over 10 s, it runs the PoW consensus mechanism to win the right to publish the
We ignore the fork problem of block FL and set up a small difficulty for PoW, which, on average, costs miner 5 s to solve a hash cryptography problem. The average time consumption per 100 iterations for the four FLs is shown in Table II.

2) Analysis of DAG-FL Efficiency:

We first evaluate the test accuracy and training loss of DAG-FL compared with other FL systems in on-device FL scenarios. As DAG-FL is an asynchronous FL empowered by DAG-based blockchain, it can well address issues of device asynchrony and has a good work efficiency.

Fig. 5 displays the test accuracy and training loss of the four FLs on both CNN and LSTM tasks. For the CNN task, DAG-FL and block FL have very close convergence rates. The target models of these two FLs achieve lower accuracy than Google FL and asynchronous FL at the first 5000 iterations. This is because Google FL and asynchronous FL can promise an equal frequency for data on each node to be used. For the LSTM task, at the first 5000 iterations, asynchronous FL and DAG-FL have better convergence rates than Google FL and block FL. Moreover, asynchronous FL performs better than DAG-FL because it ensures the average use of data. Considering the iteration latency of the four FLs in Table II, DAG-FL and asynchronous FL can complete much more FL iterations than Google FL and block FL during the same time. Due to the device asynchrony, Google FL and block FL have to waste time synchronizing the global model periodically, which misses many opportunities for nodes to do FL iterations. When the time comes to 10000 s, in the CNN task, all the four FLs can train a target model that achieves an accuracy of around 0.982. In contrast, DAG-FL and asynchronous FL can get target models with an accuracy of 0.552, while the other two FLs can only get final target models with an accuracy below 0.548 in the LSTM task.

Through the comprehensive consideration of the convergence rate and average iteration latency, we conclude that DAG-FL can well meet the challenges of device asynchrony, which is an efficient FL system with good convergence in an on-device FL scenario.

3) Discussion on the Immunity of Abnormal Nodes’ Influence:

As DAG-FL is a distributed system, it is hard to guarantee that all of the participating nodes are honest. Thus, by setting different numbers of nodes as lazy nodes, poisoning nodes, and backdoor nodes, we can get the robustness of DAG-FL in terms of accuracy performance. For the CNN task, the accuracy of different numbers of abnormal nodes is shown in Fig. 6.

In Fig. 6(a), with the number of lazy nodes increasing, the convergence rate of DAG-FL only slightly reduces between 500 and 2000 iterations. Even 30% of nodes are set to be lazy, DAG-FL still performs well as if no abnormal nodes are included. In Fig. 6(b), DAG-FL performs well when 10% or fewer nodes are set as poisoning nodes. When 20% nodes are set as poisoning nodes, DAG-FL still achieves a good
convergence at 3000 iterations. However, when 30% of nodes are set as poisoning nodes, the convergence rate of DAG-FL decreases significantly, and the target model at 1000 iterations achieves an accuracy of 0.67, which is much lower than that without abnormal nodes. This indicates that DAG-FL has been severely impacted by poisoning nodes and is going to collapse if more poisoning nodes are included, which can be treated as the threshold of working properly. As for the case that backdoor nodes are included in Fig. 6(c), we can see that DAG-FL performs good robustness unless 30% nodes are set to do backdoor attacks. When 30 backdoor nodes are included, the convergence rate of DAG-FL sharply falls down in the first 2000 iterations, which indicates that DAG-FL cannot ignore the impact of backdoor nodes anymore. The results in Fig. 6 confirm the inference that DAG-FL is insensitive to the impact of malicious activities and has good robustness when a small number of abnormal nodes are involved.

The accuracy of different FLs with 20% lazy nodes for CNN and LSTM tasks is shown in Fig. 7 and Fig. 8. We find that a small number of lazy nodes in Google FL and asynchronous FL can help the global model avoid sharp updating during the FL process. Thus, these two FLs still have good convergence rates when 20% lazy nodes are involved for both CNN and LSTM tasks. It can be observed that block FL is significantly affected by the participation of lazy nodes in terms of the convergence rate and accuracy. Compared with the accuracy at 5000 iterations in the ideal case, block FL loses about 0.02 accuracy in the CNN task and about 0.04 accuracy in the LSTM task when 20 lazy nodes are involved. The reason is that, with more associated lazy nodes, the miner in block FL usually has more time to run PoW to win the right to publish the next block. This means that more transactions published by normal nodes are dropped instead of being adopted for the global model updating.

We demonstrate the accuracy curves of different FLs with 20% poisoning nodes in Fig. 9 for the CNN task and Fig. 10 for the LSTM task. We can see that the accuracy of Google FL and asynchronous FL significantly reduces when poisoning nodes are involved; this is due to the fact that Google FL and asynchronous FL are not capable of distinguishing poisoning nodes. In contrast, these two FLs can only use the average algorithm to dilute the harmful model parameters uploaded by poisoning nodes, so as to achieve the purpose of mitigation. Since there are multiple miners to verify the model parameters uploaded by the associated nodes, the accuracy of block FL does not decline much when poisoning nodes participate. DAG-FL performs best among the four FLs when poisoning nodes are involved in both CNN and LSTM tasks. The reason is that, in the process of FL of DAG-FL, each node will validate the published transactions through the unique voting
Table III

| System      | Backdoor nodes | Attack success rate |
|-------------|----------------|---------------------|
| DAG-FL      | 5              | 0.0060              |
|             | 10             | 0.3358              |
| Block FL    | 20             | 0.6243              |
| Google FL   | 20             | 0.9166              |
| Asynchronous FL | 20       | 0.9211              |

Fig. 11. Accuracy of different FL systems with backdoor nodes for the CNN task.

4) Detection of Abnormal Nodes: In the process of DAG-FL, we regard the transactions with less than or equal to $m$ approvals as isolated transactions that make no contribution to the FL; otherwise, the transactions that exceed $m$ approvals can contribute. The contribution rate for a node in DAG-FL is the ratio of the number of contributing transactions to the number of all transactions published by the node. By comparing the contribution rates of nodes in DAG-FL, we can evaluate the ability of DAG-FL to do anomaly detection during the FL process. Table IV shows the contribution rates of DAG-FL in different scenarios with abnormal nodes after running for 10000 s, in which $r$ is the average contribution rate of all nodes and $r_0$ is the average contribution rate of abnormal nodes.

For both CNN and LSTM tasks, it seems that lazy nodes cannot be distinguished in DAG-FL by comparing contribution rates when setting $m$ as 0. Through our analysis, the reason is that, though the transactions published by lazy nodes cause worse staleness, most of the transactions are not isolated due to fewer approvals. By setting $m = 1$, DAG-FL seems to have a good ability of anomaly detection when lazy nodes are few.

For poisoning nodes, by comparing the contribution rates for both CNN and LSTM tasks, we draw a conclusion that DAG-FL can easily distinguish the poisoning nodes during the FL process. However, with the increase in poisoning nodes, the average contribution rate of them increases accordingly, which means that the ability of poisoning nodes detection of DAG-FL is degraded.

For backdoor nodes, we find that DAG-FL has an excellent ability to detect when the number is small. However, this ability falls off a cliff when the number of backdoor nodes increases. This is because the backdoor nodes can launch a joint attack with the number increasing. Backdoor nodes prefer to approve transactions published by other backdoor nodes, which reduces the possibility of transactions published by backdoor nodes being isolated. Thus, this joint attack can slash the anomaly detection ability of DAG-FL and succeed in creating a back door in the final target model of FL.

Table IV

| Task     | Abnormal type | $m$ | Nodes | $r_0$ | $r$ | $r_0/r$ |
|----------|---------------|-----|-------|-------|-----|--------|
| CNN      | Poisoning     | 0   | 5     | 0.779 | 0.827 | 0.941  |
|          |               | 10  | 0.788 | 0.827 | 0.952 |
|          |               | 20  | 0.783 | 0.817 | 0.958 |
| LSTM     | Lazy          | 0   | 0.437 | 0.560 | 0.780 |
|          |               | 10  | 0.461 | 0.556 | 0.830 |
|          |               | 20  | 0.520 | 0.556 | 0.936 |
|          | Poisoning     | 0   | 0.178 | 0.809 | 0.221 |
|          |               | 10  | 0.352 | 0.796 | 0.442 |
|          |               | 20  | 0.615 | 0.822 | 0.749 |
|          | Lazy          | 0   | 0.033 | 0.555 | 0.059 |
|          |               | 10  | 0.004 | 0.551 | 0.015 |
|          |               | 20  | 0.277 | 0.562 | 0.493 |
|          | Poisoning     | 0   | 0.214 | 0.808 | 0.264 |
|          |               | 10  | 0.307 | 0.814 | 0.323 |
|          |               | 20  | 0.771 | 0.810 | 0.928 |
|          | Lazy          | 0   | 0.050 | 0.568 | 0.089 |
|          |               | 10  | 0.220 | 0.553 | 0.398 |
|          |               | 20  | 0.461 | 0.561 | 0.821 |
|          | Poisoning     | 0   | 0.742 | 0.827 | 0.897 |
|          |               | 10  | 0.777 | 0.829 | 0.937 |
|          |               | 20  | 0.793 | 0.830 | 0.945 |
|          | Lazy          | 0   | 0.406 | 0.560 | 0.726 |
|          |               | 10  | 0.453 | 0.560 | 0.812 |
|          |               | 20  | 0.519 | 0.569 | 0.913 |
|          | Poisoning     | 0   | 0.237 | 0.793 | 0.299 |
|          |               | 10  | 0.418 | 0.789 | 0.530 |
|          |               | 20  | 0.635 | 0.821 | 0.778 |
|          | Lazy          | 0   | 0.029 | 0.557 | 0.051 |
|          |               | 10  | 0.090 | 0.546 | 0.166 |
|          |               | 20  | 0.248 | 0.539 | 0.448 |

B. Testbed

In order to illustrate the feasibility of DAG-FL, a practical application program is implemented to achieve DAG-FL on a real testbed with five cloud nodes and a host server, as shown in Fig. 12. In the testbed, five nodes supported by the Alibaba Cloud Computing Company act as mobile devices with similar computing capacity and high network bandwidth, and a host
server acts as the external agent \( E \) with a host program running DAG-FL controlling algorithm.

The goal of the host server includes: 1) initialize all other five participating nodes; 2) maintain a local DAG; and 3) regularly compute a global model from the local DAG to observe whether the machine learning model coconstruction is completed. If the host server finds that FL is completed, it will send a termination signal to all nodes that are participated in DAG-FL. For the other five nodes, they execute the client program to run the DAG-FL updating algorithm, which can maintain the local DAG by communicating with adjacent nodes periodically and actively participate in iterations of FL when nodes are in the idle state. In Fig. 13, compared with training CNN on a single node, our DAG-FL deploying on five nodes has almost the same model convergence rate as the local CNN model in the first 50 iterations. This is because temporarily constructed global models differ a lot from each other in the early stage. In the middle stage, different global models tend to integrate with each other, resulting in a sharp increase in the convergence rate of DAG-FL. Moreover, DAG-FL on five nodes has more training data to build the CNN model, so as to achieve higher accuracy. In such an implementation experiment, we can conclude that DAG-FL can operate effectively in real situations and achieve good performance when building a CNN model.

**VI. Discussion**

In this article, we propose a DAG-FL framework to solve the problems of device asynchrony and anomaly detection, which is the first attempt to systematically apply DAG ledger technology to FL. In this section, we will discuss how to extend DAG-FL and highlight several problems to be solved in DAG-FL as future work in three aspects.

**A. Model Validation**

The model validation method of DAG-FL consensus in this article is using a small test set to compute the prediction accuracy directly, which may not be applicable in certain FL scenarios [28]. We believe that using a more advanced model validation method can enhance the immune ability of DAG-FL to abnormal nodes, reduce the delay of an iteration of DAG-FL, and improve the overall system efficiency. For example, use a pretrained autoencoder model like [29] to detect abnormal transactions when validating during an iteration of DAG-FL.

**B. Credit Evaluation**

In DAG-FL, abnormal nodes can be detected by computing the contribution rate, and the nodes with a low contribution
in DAG-FL should be punished during the process of FL. In order to avoid abnormal nodes deteriorating the system, it is feasible to design a confidence algorithm for tips’ selection based on the contribution rate of each node, in which the tips generated by normal nodes could be selected with a higher probability. In contrast, abnormal nodes with low credit could be identified, and thus, the corresponding tips would be selected for validation rarely or orphaned finally. In this way, abnormal nodes are punished to be isolated, and normal nodes can get more potential rewards.

C. Weighted Aggregation

In this article, global models are constructed by tips using the FederatedAveraging algorithm with the same weight coefficient. However, when constructing a global model to train in an iteration of DAG-FL, even tips published by normal nodes are not equally important under an asynchronous environment, where some tips containing local models with better staleness and prediction accuracy are better due to the capability of the node. Therefore, a weight aggregation algorithm like [30] can be designed to assign a larger weight coefficient for the higher quality of the local model when constructing the global model. In this way, the constructed global model can better reflect the real progress of FL at the current time to improve the training efficiency of the whole FL and get a better target model finally.

VII. RELATED WORK

Currently, studies on FL have been received considerable attention to deal with the challenges of security and privacy. In this section, we review some state-of-art works of FL in four primary aspects: synchronous FL, pseudoasynchronous FL, asynchronous FL, and blockchained FL.

A. Synchronous FL

Google FL [1] and other similar FL systems [4], [31], [32] are based on synchronous updating mechanism for FL. These systems usually allow the central server to maintain a unique global model and assign FL tasks to some nodes that are in the idle state in each round of FL. Before beginning another new round, the central server should wait for every selected node to complete its assigned FL tasks. This synchronous updating method promises a simple structure for FL and is easy to implement, but errors on any node can block synchronous FL, resulting in poor tolerance of fault. In addition, the central server needs to upload and download models frequently and is also responsible for maintaining the entire network structure of FL. Thus, the scalability of synchronous FL is limited by the communication bandwidth and computing capacity of the central server [7].

B. Pseudoasynchronous FL

Based on synchronous FL, researchers construct the pseudoasynchronous FL to allow nodes to do FL iterations freely. Nodes in a pseudoasynchronous FL system can download the global model from the central server at any time and immediately upload their trained local models to a cache. The central server regularly updates its global model by aggregating trained local models from the latest global models in the cache. In order to improve the system efficiency, the interval between the global model updating should be set small, and bottleneck nodes usually cannot upload their trained local models in time. This causes meaningful data on bottleneck nodes to be hard to utilize, which violates the original intention of making full use of the data on each node in FL. By redesigning nodes selection and global model aggregation methods, the authors in [19] have established a pseudoasynchronous FL system that can utilize data on bottleneck nodes. The central server in [19] selects the local models trained from both the latest and outdated global models for aggregation when updating the global model. However, the aggregation method and the interval between the global model updating should be well designed for different FL scenarios.

C. Asynchronous FL

The authors in [7] demonstrate that the asynchronous scheme is an attractive approach to mitigate stragglers in heterogeneous environments. As FL is more likely to be deployed in mobile wireless networks that are usually heterogeneous environments in the future, some studies on designing the asynchronous FL systems have been investigated [33], [34]. In asynchronous FL, a node can download a global model from a central server and train a local model to upload in the idle state at any time. In [10], the authors propose an asynchronous decentralized algorithm based on stochastic gradient descent. In a heterogeneous environment, the proposed algorithm is robust to build a deep learning model, such as VGG [35] and ResNet-20 [36], and can achieve efficient communications with the best-possible convergence rate. Based on the work of [10], the authors in [9] propose an asynchronous FL algorithm with an aim of weighting different local models of staleness to update the global model. Meanwhile, considering both strongly and nonstrongly convex problems, this work also proves that the asynchronous updating method can reach near-linear convergence.

D. Blockchained FL

It is noted that FL and blockchain technology are both deployed in a distributed network; many researchers attempt to combine the advantages of them and have proposed some influential blockchained FL systems [8], [16], [37]. Combining blockchain and FL, the authors in [38] propose a DeepChain architecture. This work focuses on solving the problems of privacy and audibility by storing the local machine learning model in blockchain and regarding these models as transactions, which uses the security and immunity of blockchain to prevent nodes without permission from accessing information stored in transactions. Similarly, in order to store sensitive and personal data in a diskless environment, studies, such as [39], adopt blockchain to form a data-sharing platform, which is beneficial for data privacy and security. On this basis, the authors in [40] utilize a sharding-based blockchain protocol to well protect the convergence of FL when large-scale nodes are involved. Furthermore, researchers in [41] also propose a
sharding-based FL system by applying a two-layer blockchain framework.

In recent works [3], [15], researchers are interested in a deep combination of blockchain and FL and get more suitable models for on-device scenarios or others. Both of the works use the miner in blockchain to complete the task of the central server in traditional FL. The authors in [3] design a block FL with multiple miners to coordinate FL tasks and maintain the global model, and the procedure can be described as follows: 1) the node downloads the global model from its associated miner and get a trained local model; 2) the trained local model will be uploaded to the associated miner as a transaction; and 3) the miner will confirm the validity of the uploaded transaction, and the confirmed transactions will be stored in the candidate block of the miner. Once candidate blocks collect enough transactions or wait for a certain time, all miners enter the consensus stage together and run PoW to compete for a winner to publish its own candidate block on the blockchain. Besides, miners are capable of allocating rewards when publishing their blocks on blockchain to motivate mobile devices to participate in FL. Followed up the work [3], the authors in [15] propose an FLchain architecture that considers MEC servers as miners to solve problems brought by central servers in traditional FLs. The FL systems in these two works are more like the pseudosynchronous ones, in which nodes are freed after uploading local trained models to miners. Compared with only using blockchain to store and transmit information, works, such as [3], [15], provide a feasible way to achieve efficient fusion of blockchain and FL in terms of data security and privacy.

However, these previous works still remain issues to be addressed on both blockchain and FL. On the one hand, these works insist on the PoX consensus mechanism, which naturally leads to the pseudosynchronous framework that cannot meet the device asynchrony in on-device FL. Besides, the introduced miners would generate extra delay and resource consumption on consensus but not for FL at all. On the other hand, these works rely on miners to validate uploaded local models stored in transactions, which might cause miner’s dilemmas due to the conflict of interest between blockchain and FL. As the transaction validation is executed on miners, the target model might be trained toward the expected direction of miners instead of nodes, resulting in the problem of local optima.

To this end, for asynchronous bookkeeping and low resource consumption of DAG ledger technology, this article proposes DAG-FL introducing DAG empowers asynchronous FL framework. Through the workload assign of consensus and model validation on each node in a decentralized manner, miners and mining are not needed anymore. By combining the voting mechanism of DAG ledger technology with the process of local model validation in FL, DAG-FL can detect abnormal nodes and mitigate their impact autonomously to guarantee security. Meanwhile, as the model validation is operated on each node without professional miners, the target model of FL would be trained under the observation of all nodes avoiding the training deviation for any reason.

VIII. Conclusion

In this article, we proposed a DAG empowered FL system named DAG-FL, which combines DAG ledger technology to overcome the problems of mobile device asynchrony and abnormal nodes to complete the model coconstruction task of FL efficiently. DAG-FL is constructed by a three-layer asynchronous architecture, including FL, DAG, and application layers taking the responsibilities of model training, communicating, and observing. In this manner, the DAG layer could provide blockchain as a service (BaaS) establishing DAG-FL controlling and DAG-FL updating algorithms for the operation process of DAG-FL. Meanwhile, to maintain the stability and efficiency of DAG-FL, we discussed two deployment details explaining how to keep a reasonable number of tips on DAG and how to address the staleness of FL results. Finally, experimental results showed that DAG-FL has a higher system efficiency and a better target model with device asynchrony compared with the other three benchmark FL systems. Meanwhile, DAG-FL is also proved to be insensitive to the impact of abnormal nodes and has the ability of anomaly detection. Moreover, the implementation results on the testbed prove that DAG-FL can achieve high stability and efficiency as expected. To provide some open discussion issues to extend DAG-FL, we highlighted the concerns on model validation, credit evaluation, and weighted aggregation as the possible future direction.

REFERENCES

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, “Communication-efficient learning of deep networks from decentralized data,” in Proc. 20th Int. Conf. Artif. Intell. Statist., vol. 54, Fort Lauderdale, FL, USA: PMLR, Apr. 2017, pp. 1273–1282.
[2] B. McMahan and D. Ramage, “Federated learning: Collaborative machine learning without centralized training data,” Google Res. Blog, vol. 3, Apr. 2017.
[3] H. Kim, J. Park, M. Bennis, and S.-L. Kim, “Blockchained on-device federated learning,” IEEE Commun. Lett., vol. 24, no. 6, pp. 1279–1283, Jun. 2020.
[4] A. Hard et al., “Federated learning for mobile keyboard prediction,” 2018, arXiv:1811.03604. [Online]. Available: http://arxiv.org/abs/1811.03604
[5] S. Guo, Y. Qi, Y. Jin, W. Li, X. Qiu, and L. Meng, “Endogenously trusted DRL-based service function chain orchestration for IoT,” IEEE Trans. Comput., early access, Jan. 18, 2021, doi: 10.1109/TC.2021.3051681.
[6] L. Zhang, B. Cao, Y. Li, M. Peng, and G. Feng, “A multi-stage stochastic programming-based offloading policy for fog enabled IoT-Ehealth,” IEEE J. Sel. Areas Commun., vol. 39, no. 2, pp. 411–425, Feb. 2021.
[7] 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.
[8] M. H. U. Rehman, K. Salah, E. Damiani, and D. Svetinovic, “Towards blockchain-based reputation-aware federated learning,” in Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS), Jul. 2020, pp. 183–188.
[9] C. Xie, S. Koyejo, and I. Gupta, “Asynchronous federated optimization,” 2019, arXiv:1903.03934. [Online]. Available: http://arxiv.org/abs/1903.03934
[10] X. Lian, W. Zhang, C. Zhang, and J. Liu, “Asynchronous decentralized parallel stochastic gradient descent,” in Proc. 35th Int. Conf. Mach. Learn., vol. 80. Stockholm, Sweden: PMLR, Jul. 2018, pp. 3043–3052.
[11] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, “How to backdoor federated learning,” in Proc. Int. Conf. Artif. Intell. Statist., vol. 108, Aug. 2020, pp. 2938–2948.
[12] C. Xie, O. Koyejo, and I. Gupta, “SLSGD: Secure and efficient distributed on-device machine learning,” in Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases, Cham, Switzerland: Springer, 2019, pp. 213–228.
