A Blockchain System for Clustered Federated Learning with Peer-to-Peer Knowledge Transfer

Honghu Wu*
State Key Laboratory for Novel Software Technology
Nanjing University, China
hhwu.nju@gmail.com

Xiangrong Zhu*
State Key Laboratory for Novel Software Technology
Nanjing University, China
xrzhu.nju@gmail.com

Wei Hu†
State Key Laboratory for Novel Software Technology
Nanjing University, China
whu@nju.edu.cn

ABSTRACT

Federated Learning (FL) is a novel distributed, privacy-preserving machine learning paradigm. Conventional FL suffers from drawbacks such as single point of failure and client drift. Blockchain is a distributed computing architecture famous for decentralization, transparency, and traceability. Incorporating blockchain as the underlying basis for FL decentralizes the FL process and brings opportunities to resolve the drawbacks. However, there still remain challenges to fulfilling FL with blockchain, regarding effectiveness, efficiency, and security. In this paper, we propose a new blockchain system for FL, called FedChain. To mitigate client drift and accelerate training, we present a clustered semi-asynchronous method for model aggregation. To optimize the local training in FL, we introduce a knowledge transfer method using other clients on the peer-to-peer network of blockchain. Moreover, we implement an access control mechanism to store and transmit models safely and efficiently. Extensive experiments on various benchmark datasets show that FedChain achieves superior results in accuracy, convergence, throughput, and latency.

PVLDB Reference Format:
Honghu Wu, Xiangrong Zhu, and Wei Hu. A Blockchain System for Clustered Federated Learning with P2P Knowledge Transfer. PVLDB, 17(5): 966 - 979, 2024.
doi:10.14778/3641204.3641208

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/nju-websoft/FedChain.

1 INTRODUCTION

Federated Learning (FL) is a novel distributed, privacy-preserving machine learning paradigm [30], where multiple clients with local data work together for improved model performance without raw data sharing. Currently, FL has been widely employed in various privacy-sensitive applications such as financial services [50], smart healthcare [32], and recommender systems [49].

In a common FL framework [4], a central server coordinates all clients for aggregating local models. It often suffers from several inherent limitations. *First*, the client-server architecture of FL depends highly on the single central server, which raises the risk of a single point of failure (SPOF). *Second*, mainstream FL frameworks aggregate the local models of all clients to learn a global model fitting different local data. However, the heterogeneity in local data may lead to inconsistent optimization directions of the local models, slow convergence of the global model, and unsatisfied performance on local datasets, which is referred to as *client drift* [17]. *Third*, FL follows a data-parallel distributed training strategy [51], either synchronous [43] or asynchronous [47]. Synchronous FL means that all clients train with the same model parameters. The server awaits clients to complete the mini-batch training and performs the aggregation of local models and the update of the global model. In the case of imbalanced computing power or communication speed, there would be a serious short-board effect, that is, a straggler slows down the overall process. In asynchronous FL, the clients update model parameters separately without waiting for each other. Although the asynchronous strategy seems more efficient, it raises the problem of gradient expiration during training, which may result in unstable convergence or a suboptimal solution.

Blockchain is a novel distributed storage architecture and computing paradigm famous for decentralization, openness and transparency, traceability and tamper resistance. Many blockchain systems [33, 34, 48] have been deployed in e-commerce, security trading, IoT, etc. Incorporating blockchain as the underlying architecture for FL decentralizes the FL process and enhances it in the aforementioned aspects.

First, as a replicated distributed storage system, blockchain ensures data consistency through the consensus mechanism [13]. The characteristic of decentralized consistency would eliminate the dependence of FL on the single central server and reduce the risk of SPOF. Moreover, the updated models can be recorded and traced on blockchain, and whether a model is maliciously tampered with can be verified. These are difficult to be realized in traditional FL.

Second, the blockchain network follows the peer-to-peer (P2P) architecture, where peers are connected to each other. Traditional FL generally adopts the client-server architecture, which relies on bidirectional communication between the server and clients to transmit model parameters and gradient updates. Through the P2P architecture, the clients can learn from similar models flexibly to mitigate the impact of data heterogeneity.

Third, blockchain offers an immutable, verifiable, and traceable distributed storage solution to FL. Under the protection of

*The two authors contributed equally to this work.
†Wei Hu is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 17, No. 5 ISSN 2150-8097.
doi:10.14778/3641204.3641208

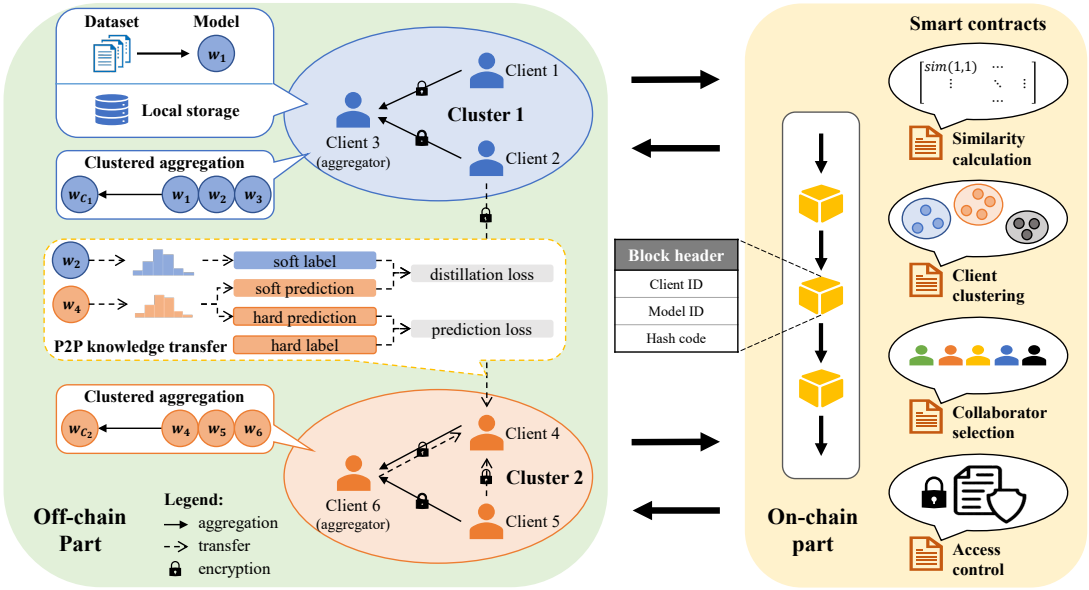

Figure 1: System architecture of FedChain

blockchain, there can be more than one trusted aggregator, and the clients can access reliable models in parallel. It brings the probability of combining synchronous and asynchronous FL to achieve a balance between time efficiency and model accuracy.

In this paper, we propose a blockchain system for FL, dubbed *FedChain*, which innovatively improves the global aggregation and local training of traditional FL. FedChain focuses on the following two key challenges to fulfilling FL with blockchain.

Challenge 1. *What is a good architecture to combine blockchain and FL?* As far as we know, a few works [35] design blockchain-based FL systems to handle SPOF. However, they are a naive combination of blockchain and FL without considering the improvement of the training process. These works mainly focus on the security and privacy of FL systems. But simply using blockchain to record information cannot solve other inherent drawbacks of FL mentioned above. Our goal is to strengthen the correlation between blockchain and the training process of FL. This demands a new architecture to deeply integrate blockchain into various phases of FL such as global aggregation, local training, and model management.

Challenge 2. *How to improve the performance of FL with blockchain while not bringing much extra burden?* The P2P network of blockchain naturally provides more channels for data exchange. It can break the isolation between clients in traditional FL and raise the probability of improving the accuracy of models. However, the naive combination of blockchain and FL does not solve the problem of synchronous waiting in traditional FL. Furthermore, as the parameter size of large models is much larger than the general block size (e.g., the regular block size threshold of Hyperledger Fabric is 64MB [1]), uploading models to blockchain would cause a significant communication overhead. On one hand, we introduce a dynamic clustered FL method in this paper. We implement smart contracts to divide clients into groups for reducing idle time (i.e., the time from local model update to global model synchronization). On the other hand, we leverage the P2P network to transfer knowledge among clients during the local training phase to alleviate the poor performance caused by data heterogeneity.

The main contributions of this paper are outlined as follows:

- We propose a novel FL architecture built upon blockchain. FedChain couples blockchain and FL in depth by fully decentralizing the training process of FL, making the strengths of blockchain better applicable to FL. (Sect. 3)
- We introduce a clustered semi-asynchronous aggregation method. Clients are divided into different clusters through smart contracts, and model aggregation is constrained to the clients in each cluster. This mitigates the heterogeneity in aggregation and accelerates training. (Sect. 4)
- We present a P2P knowledge transfer method for enhancing the local training in FL. We formulate the collaborative client selection problem and leverage smart contracts to find solutions with a guaranteed approximation ratio. Clients distill knowledge from collaborators to improve the performance on imbalanced data. (Sect. 5)
- With a general threat model, we implement FedChain based on an efficient and secure model storage and access control mechanism, defending against potential attacks. (Sect. 6)
- We carry out extensive experiments on different types of benchmark datasets. The experimental results show the superiority of FedChain in accuracy, convergence, throughput, and latency against state-of-the-art baselines. (Sect. 7)

2 PRELIMINARIES

2.1 Federated Learning

With the explosive growth of smart devices and rising concerns about data privacy, FL becomes one of the most popular distributed learning paradigms. As a pioneering work, FedAvg [30] consists of

three steps: model initialization, local training, and global aggregation. A server first initializes model parameters and distributes them to all clients. After receiving the global model, each client trains a local model with its local data and returns it to the server. The server collects all the local updates and aggregates them into the global model. After a few rounds of communication, the converged global model can achieve similar performance to the model centrally trained on all local datasets. The vanilla FedAvg suffers badly from the data and device heterogeneity in real applications.

The data heterogeneity problem in FL is mainly due to the fact that the datasets on local clients are independently distributed and do not follow the same sampling strategy, a.k.a., non-independent and identically distributed (abbr. non-IID). Data heterogeneity leads to inconsistent gradient directions during local training and poor performance of the global model on local datasets. FedProx [26] and MOON [25] attempt to constrain the optimization directions of the local models via regularization and contrastive loss, respectively, to achieve a converged global model faster. However, this may lead to underfitting over local datasets. FedChain enhances the specificity of federated models and avoids the local optimum by dynamically dividing clients into clusters based on data features.

As clients vary in storage, computation, and communication capabilities, device heterogeneity is another pressing problem in FL. As with FedAvg, a server awaits all clients to train and upload the local updates before aggregation, resulting in a significant overall delay, particularly when facing SPOF. A number of *asynchronous* FL algorithms [12, 47] are proposed to cope with device heterogeneity. FedAsync [47] adopts an asynchronous communication protocol with an adaptive learning rate. However, fully asynchronous communication leads to inconsistency of model parameters and slow convergence of the global model. *Semi-asynchronous* FL [7, 54] combines synchronous and asynchronous FL to mitigate the influence of stragglers. FedAT [7] classifies clients into logical tiers based on response latency and combines synchronous intra-tier training and asynchronous cross-tier training to minimize the straggler effect. *Clustered* FL [10, 24, 37, 54] groups clients with similar data distributions, computation capabilities, etc. FedHiSyn [24] presents a hierarchical synchronous FL framework, which clusters devices in terms of computing capacity and leverages a ring topology for communication among devices. In this paper, we design a clustered semi-asynchronous method, which clusters clients based on both data distribution and training latency. Synchronous FL is performed within clusters to ensure performance and asynchronous update is performed across clusters to ensure efficiency.

For P2P FL, BrainTorrent [36] is a decentralized framework designed for medical applications, which enables collaborative training among medical centers without a central server. The work [22] defines FL as social learning on a graph and gives high probability guarantees on the minimal number of training samples to learn the optimal global model. In its P2P learning, clients take a Bayesian-like method via beliefs over the model parameter space and update the beliefs by aggregating information from their neighbors. To resolve client drift caused by data heterogeneity and model averaging, Def-KT [23] performs decentralized FL via mutual knowledge transfer. Due to the lack of a global view to guide the general optimization direction, it is hard for these methods to achieve stable promotion in accuracy. We present a P2P knowledge transfer method to enhance the local training in FL and a multi-client selection algorithm to improve the effectiveness of collaboration among clients.

2.2 Blockchain-based Federated Learning

Blockchain, stemming from the Bitcoin network [31], is a distributed ledger managing transactions and states across the whole network. *Smart contracts* are code deployed on each blockchain client with predefined logic for data manipulation and external calls [42]. FedChain builds a blockchain system based on Hyperledger Fabric [1], enabling clients to register for FL participation. It also maintains flexibility in accommodating other blockchains such as R3 Corda [14] and FISCO BCOS [3].

Much progress has been made in combining FL with blockchain in recent years. Existing blockchain systems for FL have different design directions. In [19, 27, 55], blockchain is used as an incentive factor to promote the FL process. Rewards are assigned to clients according to their training contributions. To enhance the security of FL and protect the model privacy, blockchain acts as a distributed database [9]. Encrypted model parameters are stored and verified with blockchain. Also, a few works [29, 45] are devoted to analyzing and optimizing the computation and communication overhead of blockchain systems, in which we are particularly interested.

BlockFL [19] uploads all local updates of a federated regression model on blockchain. Clients can download local updates to aggregate a new global model. They get rewards by training local models, verifying local updates, and generating new blocks. BlockFL analyzes and assesses the end-to-end latency of FL with blockchain. BFLC [27] proposes a blockchain-based FL framework and devises a committee consensus mechanism to reduce malicious attacks and consensus computing costs. Each client is assigned a score based on the verified local model, and the score is used to elect the committee. Honest clients constitute the committee in charge of the verification of local models. ScaleSFL [29] designs a sharding solution with shard-level consensus and mainchain consensus for blockchain-based FL. Clients are assigned to different shards and each shard generates a local shard-aggregated model with shard-level consensus. The verified shard-aggregated models are coordinated for global aggregation with mainchain consensus.

In summary, existing works on blockchain-based FL suffer from three drawbacks. *First*, involving blockchain brings additional processes like verification, auditing, and assignment, which leads to extra computation costs compared with traditional FL. *Second*, to achieve consistency among distributed clients, uploading model parameters to blockchain goes through the consensus process, which is a performance bottleneck for current systems. *Third*, to our best knowledge, little work attempts to unleash the potential of the P2P architecture of blockchain for improving the model accuracy of FL. In this paper, we go beyond the existing works with clustered semi-asynchronous aggregation and P2P knowledge transfer.

3 SYSTEM ARCHITECTURE

Fig. 1 shows the system architecture of FedChain. The P2P network on which FedChain is built coordinates the on-chain and off-chain parts and ensures communication among clients. There are three functional modules, namely *clustered FL*, *P2P knowledge transfer*, and *blockchain-based access control*. They correspond to the global

aggregation, local training, and model management in vanilla FL, respectively. The on-chain part is responsible for metadata storage and automatic execution of these modules, and the off-chain part is engaged in specific computation and storage. Multiple smart contracts are deployed on FedChain to provide key computing functions for each module.

- **Clustered FL.** We divide clients into clusters according to their data features and computing power. The clients in each cluster collaborate for one individual model, and one client is selected as the aggregator within the cluster. Learning multiple intra-cluster models instead of one single global model can alleviate the client drift problem and improve overall accuracy [37]. Furthermore, dividing clients with similar computing power for synchronization can effectively reduce idle time. The automatic execution of this module depends on the *similarity calculation smart contract* and the *client clustering smart contract*.
- **P2P knowledge transfer.** In local training, clients train their local models on private datasets. To reduce the effect of client drift, local updates are optimized collaboratively based on the P2P network of blockchain. A client improves the performance of its local model by transferring knowledge from a few other models. The collaborative clients are obtained by the *collaborator selection smart contract*.
- **Blockchain-based access control.** To ensure model privacy and reduce consensus overhead, we store all model parameters locally on clients and only upload relevant metadata on-chain. We leverage the traceability and immutability of blockchain to provide access control, enabling clients to legally and safely read the model parameters managed by other clients. Model replication is also used to balance the access workload.

The architecture design of FedChain is based on our understanding of decentralized FL. We replace the aggregation of a single central server with multiple clusters. Based on the P2P network, we transfer knowledge among clients. In addition, the traceable and immutable features ensure the security of model access and transmission. Compared to existing systems [19, 27], our architecture transfers heavy training and model storage to off-chain devices, and blockchain is responsible for core, lightweight metadata storage and contract operations. Thus, FedChain is guaranteed in terms of performance and efficiency. See the technical details below.

4 CLUSTERED FEDERATED LEARNING

We leverage blockchain for decentralizing global aggregation to avoid SPOF. Also, we cluster similar clients and aggregate models in parallel to mitigate the heterogeneity in model aggregation and accelerate the training process. In this section, we first present feature extraction, the technique employed to capture the data distributions of clients and measure their similarity to build clusters. Then, we propose a semi-asynchronous FL approach based on clusters.

4.1 Feature Extraction

Feature extraction is the basis for clustered FL. Its purpose is to generate a digest that can reflect the data distribution of each client for similarity calculation. Existing clustered FL systems [37] use the gradient direction to measure the similarity, which requires the central server to collect the gradients from all clients in each aggregation iteration. As the size of the model gradients is equal to that of the model parameters, uploading the gradients directly for similarity calculation is inefficient.

PFA [28] leverages the sparsity property of neural networks to represent the raw data in clients and group clients with similar data distributions, which achieves personalization and privacy. We borrow this idea and design a feature extraction method based on local training. As the feature map is often unique to a particular input, we use it as a *signature* to distinguish different data distributions.

Given the set of clients $\mathcal{K}$, each client $K_i \in \mathcal{K}$ has a private dataset $\mathcal{D}_i = \left\{(\mathbf{x}_h, y_h)\right\}_{h=1}^{N_i}$, where i denotes the serial number in $\mathcal{K}$, N_i represents the size of the dataset, $\mathbf{x}_h$ and y_h are the feature and label of the h-th data sample, respectively. At each local round, client K_i trains a local model w_i on $\mathcal{D}_i$. We select n kernel functions from the intermediate layers in w_i and record them in the genesis block of blockchain for clients to perform consistent feature mapping.

In the feed-forward process, client K_i extracts the feature matrix $F_k(\mathbf{x}_h) \in \mathbb{R}^{H \times W}$ from the output of the k-th selected kernel given input $\mathbf{x}_h$, where H and W are the height and width of $F_k(\mathbf{x}_h)$, respectively. We compute the k-th signature of $\mathbf{x}_h$ as follows:

$$sig_k(\mathbf{x}_h) = \frac{zeros\big(F_k(\mathbf{x}_h)\big)}{H \times W}, \tag{1}$$

where $zeros(\cdot)$ is a function counting how many zeros are there in the matrix. The signature of the k-th kernel on dataset $\mathcal{D}_i$ is calculated by averaging all sample features:

$$sig_k(\mathcal{D}_i) = \frac{1}{N_i} \sum_{h=1}^{N_i} sig_k(\mathbf{x}_h). \tag{2}$$

In this way, client K_i extracts the signatures during local training and forms a signature vector $\mathrm{S}_i = \big(sig_1(\mathcal{D}_i), sig_2(\mathcal{D}_i), \ldots, sig_n(\mathcal{D}_i)\big)$. Each client generates a signature vector reflecting the local data distribution and uploads it to blockchain.

Take VGGNet [39], a well-known image recognition model, for example. The size of the convolution kernel is 3×3 and suppose that the input data size is 224×224. The size of the generated signature does not exceed 4KB, which is much smaller than the size of the gradients 528MB, making both uploading and consensus efficient.

4.2 Similarity Calculation

FedChain divides clients into different groups according to their data distributions and training efficiency for aggregation. Here, we first consider the similarity of data distributions. The signature vector of each client is stored on-chain, and we use cosine similarity as the metric to calculate the similarity between vectors. Given the signature vectors $\mathrm{S}_i, \mathrm{S}_j$ of clients K_i, K_j, respectively, their distribution similarity is calculated as follows:

$$sim_{\text{distr}}(K_i, K_j) = \cos(\mathrm{S}_i, \mathrm{S}_j). \tag{3}$$

To divide clients with similar training efficiency into the same group, we require each client to track and upload the time consumption of local training at each local round. Given the training time sequences $(T_{i,1}, T_{i,2}, \ldots, T_{i,t_i})$ and $(T_{j,1}, T_{j,2}, \ldots, T_{j,t_j})$ of clients K_i and K_j, respectively, t_i and t_j are not necessarily equal because

there may be a greater number of iterations with faster computing capacity. We first truncate the two sequences to the same length by

$$\mathbf{T}_i = (T_{i,t_i-\min(t_i,t_j)+1}, \ldots, T_{i,t_i}), \tag{4}$$

$$\mathbf{T}_j = (T_{j,t_j-\min(t_i,t_j)+1}, \ldots, T_{j,t_j}). \tag{5}$$

For example, given $t_i = 10, t_j = 3$, we intercept the latest $\{T_{i,8}, T_{i,9}, T_{i,10}\}$ of client K_i as $\mathbf{T}_i$ and $\{T_{j,1}, T_{j,2}, T_{j,3}\}$ of client K_j as $\mathbf{T}_j$.

Then, given the above two training time sequences of equal length, the speed similarity between K_i and K_j is calculated as

$$sim_{\text{speed}}(K_i, K_j) = \cos(\mathbf{T}_i, \mathbf{T}_j). \tag{6}$$

The similarities of data distributions and training speed are combined to measure the overall similarity between clients:

$$sim(K_i, K_j) = \alpha \cdot sim_{\text{distr}}(K_i, K_j) + (1-\alpha) \cdot sim_{\text{speed}}(K_i, K_j), \tag{7}$$

where α is a hyperparameter to adjust the similarity weights.

The data involved in similarity calculation is uploaded by clients and stored on-chain. We deploy a smart contract named *similarity calculation smart contract* to complete the entire calculation process. At the current global round t, the contract calculates the similarity matrix $\mathbf{M}_t \in \mathbb{R}^{|\mathcal{K}|\times|\mathcal{K}|}$ with $\mathbf{M}_t[i][j] = sim(K_i, K_j)$ and stores it on-chain. The execution of the contract is in parallel with the local training of the clients for efficiency.

4.3 Clustered Aggregation

Fig. 2 shows a graphical comparison of three types of FL methods. For synchronous FL, each client waits for the server to finish aggregation before a new round of local training, and the waiting time depends on the straggler, e.g., Client 3 in Fig. 2(a). For asynchronous FL, every time the server receives a local model, it immediately updates the global model. This indicates that clients do not need to await updates from stragglers. However, we have already discussed the weakness of asynchronous FL in the introduction.

To improve efficiency and ensure performance, we design clustered semi-asynchronous FL in FedChain. Synchronous FL is performed within each cluster, and clusters do not affect each other. Let us see Fig. 2(c). Suppose that four clients are divided into two clusters, where Clients 1 and 2 form Cluster 1, and Clients 3 and 4 form Cluster 2. Server 1 and Server 2 are selected from clients in the two clusters for aggregation, respectively. Clients train the local models on the local datasets based on the cluster models. A server within the cluster detects a new local update and resets the timer. The server performs aggregation when enough updates are received or timeout. Clients pull the new cluster models to start the next round of local training. When more than half of the clusters submit the metadata of cluster models, clients are re-clustered into different clusters. Clients discover the change of clusters and submit the local updates to the new servers accordingly. The workflow is terminated until the clusters no longer change and cluster models reach convergence.

We implement a smart contract named *client clustering smart contract* to divide clients into clusters. It takes the similarity matrix $\mathbf{M}_t$ as input and the output is a set of clusters $\mathcal{G}_t = \{C_1, C_2, \ldots, C_{n_t}\}$, where n_t is the number of clusters. Since the number of clusters is uncertain, we leverage $\mathbf{M}_t$ to generate an agglomerative hierarchical clustering dendrogram. In this process, each client is initially a cluster, and the closest clusters are merged by calculating

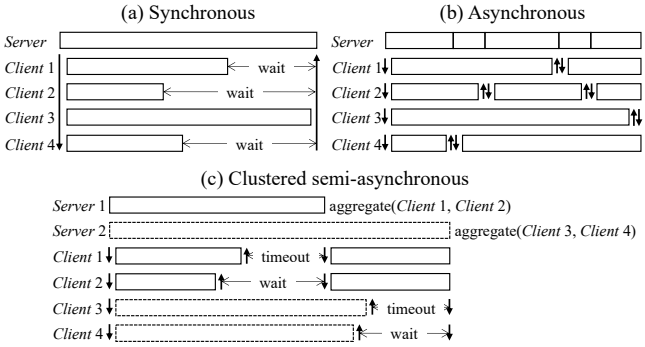

Figure 2: Synchronous, asynchronous, and clustered semi-asynchronous FL

the distance between the clusters until one large cluster remains. We employ the average linkage [18] to measure the distance for reducing the influence of outliers. For two clusters C_a and C_b, we compute the average distance of all client pairs as follows:

$$sim_{\text{avg}}(C_a, C_b) = \frac{1}{|C_a| \cdot |C_b|} \sum_{K_i \in C_a} \sum_{K_j \in C_b} sim(K_i, K_j). \tag{8}$$

The time complexity of the clustering process is $O(|\mathcal{K}|^3)$. Since the clustering process is in parallel to the local training of clients, it does not cause an efficiency bottleneck in FedChain.

Although we divide clients with similar efficiency into the same cluster, the aggregation process within the cluster is still synchronous. The timeout mechanism avoids continuously awaiting stragglers, who may upload local updates optimized based on earlier cluster models. Directly adopting the stale updates slows down the convergence [46], while discarding them wastes training efforts. To properly utilize stale updates without affecting convergence, we propose a freshness-based aggregation method. Given the local round number set $\{t_j \mid K_j \in C_a\}$ within cluster C_a, the freshness of the updated local model w_i from client $K_i \in C_a$ is calculated by

$$fresh(w_i) = \left(\max_{K_j \in C_a} (t_j - t_i + 1) \right)^{-\mu}, \tag{9}$$

where μ is the hyperparameter for adjusting the freshness value.

With the freshness as weight, the aggregator generates the cluster model w_{C_a} as follows:

$$w_{C_a} = \sum_{K_i \in C_a} \frac{fresh(w_i)}{\sum_{K_j \in C_a} fresh(w_j)} w_i. \tag{10}$$

FedChain produces a set of cluster models, denoted by $\mathcal{W} = \{w_{C_1}, w_{C_2}, \ldots, w_{C_{n_t}}\}$. Within the cluster C_a, client K_i pulls the cluster model w_{C_a} for local training. After the re-clustering process is completed, client K_i may be divided into another cluster C_b. As a result, the local model w_i updated based on the previous cluster model w_{C_a} is uploaded to the aggregator of C_b for aggregation.

5 PEER-TO-PEER KNOWLEDGE TRANSFER

The conventional FL architecture [26, 30] pays much attention to the client-server connection. Forwarding messages from a client to other clients through the server would impose a heavy traffic burden on the server. Model exchange between clients is considered in

a few existing works on P2P FL [22, 23, 36]. Benefiting from the P2P network of blockchain, clients can communicate with each other without extra burden. Furthermore, transferring knowledge from a global scope can prevent the local optimum caused by intra-cluster aggregation. In this section, we present a knowledge distillation method to improve the effectiveness of local training.

5.1 Collaborative Client Selection

Previous studies [23] randomly pick clients as teachers for knowledge transfer. However, the randomly selected teachers may perform poorly due to large divergence in data features. To improve the performance of the local model for client K_i, we seek to select informative clients as teachers. Furthermore, compared with a single teacher [15], multiple teachers can provide richer knowledge to enhance the generalization of the model and help avoid overfitting and resist noise [52]. In this paper, in addition to multiple teachers, we propose two constraints for the selection policy: (1) clients holding stale models should not be selected as they may lead to performance degradation; (2) the probability of repeatedly selecting the same client should be reduced as this would cause overfitting due to excessive distillation. Together, we define the collaborative client selection problem for client K_i as follows:

$$\max_{\mathcal{K}^* \subseteq \mathcal{K}} \sum_{K_j \in \mathcal{K}^*} sim_{\text{distr}}(K_i, K_j), \tag{11}$$

$$\text{s.t.} \sum_{K_j \in \mathcal{K}^*} L_{i,j} \leq \psi_i, \tag{12}$$

$$t_j \geq t - \lambda, \quad \forall K_j \in \mathcal{K}^*. \tag{13}$$

The objective is to maximize the overall similarity of data distributions between K_i and other clients in $\mathcal{K}^*$. In the *first* constraint, ψ_i denotes the cost threshold for client K_i. $\mathbf{L}_i = \{L_{i,1}, L_{i,2}, \ldots, L_{i,|\mathcal{K}|}\}$ is a cost vector of length $|\mathcal{K}|$ maintained by client K_i. The value $L_{i,j}$ indicates the cost of selecting client K_j for knowledge distillation. Specifically, the cost $L_{i,j}$ is calculated as follows:

$$L_{i,j} = \frac{1}{\sum_{(\mathbf{x}_h, y_h) \in \mathcal{D}_i} KL(\mathcal{P}_{i,h}, \mathcal{P}_{j,h})}, \tag{14}$$

where $KL(\cdot, \cdot)$ is the Kullback-Leiber divergence to measure the difference between two probability distributions. $\mathcal{P}_{i,h}$ and $\mathcal{P}_{j,h}$ denote the soft labels for the h-th data sample generated by the local model w_i and the teacher model w_j, which are computed as

$$\mathcal{P}_{\cdot,h} = softmax\Big(\frac{f(\mathbf{x}_h; w_\cdot)}{\tau}\Big), \tag{15}$$

where $f(\cdot)$ is the predicted probability distribution over the classes and τ is the temperature of distillation [15]. The *second* constraint ensures that the model of client $K_j \in \mathcal{K}^*$ is not outdated, and λ is the freshness threshold. We do not consider sim_{speed} for knowledge distillation here, because it measures the time costs of local training, which is irrelevant to informative collaborator selection.

Theorem 5.1. *Collaborative client selection is NP-hard.*

This problem is a variation of the max-min diversification problem. More precisely, it is a max-sum diversification problem [6, 11]. Therefore, collaborative client selection is NP-hard. We solve the collaborative client selection problem in a greedy manner with Algorithm 1. The sorting step in Line 5 costs the most time. Thus, the time complexity of the algorithm is $O\big(|\mathcal{K}| \log (|\mathcal{K}|)\big)$.

Algorithm 1: Collaborator selection smart contract

Input: client set $\mathcal{K}$, requester client K_i, distribution similarities $\{sim_{\text{distr}}(K_i, K_j) \mid K_j \in \mathcal{K}\}$, cost vector $\mathbf{L}_i$, local rounds $\{t_j \mid K_j \in \mathcal{K}\}$, global round t, loss threshold ψ_i, freshness threshold λ

Output: optimal collaborative client set $\mathcal{K}^*$

1 initialize $\mathcal{K}^* \leftarrow \emptyset, \mathbf{R} \leftarrow [], U \leftarrow 0$;
2 **foreach** $K_j \in \mathcal{K}$ **do**
3 $\quad r_j \leftarrow \frac{sim_{\text{distr}}(K_i, K_j)}{L_{i,j}}$;
4 $\quad \mathbf{R}.append(r_j)$;
5 sort $\mathbf{R}$ in descending order;
6 **foreach** $r_j \in \mathbf{R}$ **do**
7 $\quad$ **if** $t_j < t - \lambda$ **then** continue; `// ignore stale models`
8 $\quad U \leftarrow U + L_{i,j}$;
9 $\quad$ **if** $U > \psi_i$ **then** break; `// exceed cost threshold`
10 $\quad \mathcal{K}^* \leftarrow \mathcal{K}^* \cup \{K_j\}$;
11 **return** $\mathcal{K}^*$;

Theorem 5.2. *Algorithm 1 gives a 2-approximation guarantee to solve the collaborative client selection problem [11].*

5.2 Multi-client Knowledge Distillation

During the local training process of vanilla FL, each client derives the local model from the global model and optimizes it with the local dataset. In FedChain, we replace the local training of client K_i by transfer learning from other models with an increasing probability p_{t_i}. Specifically, we increase p_{t_i} from an initial probability to the maximal probability with a fixed stride. There are three main reasons why the probability should grow with the rounds. The *first* reason is that the student needs to fetch the teacher model to its local storage in knowledge transfer, which incurs additional communication costs. If we replace all local training with knowledge transfer, the time cost of the system may become unaffordable. *Second*, knowledge transfer requires the teacher model to have high accuracy on the specific task so that it can transfer useful information. It is more beneficial to perform knowledge transfer as the round increases, which better improves the model performance. *Third*, as local training and knowledge transfer are both important for FedChain, clients should pay more attention to local training in the early rounds to acquire local knowledge. Formally, for client K_i in cluster C_a with its local data $\mathcal{D}_i$, the prediction loss is defined as

$$\mathcal{L}^{\text{predict}} = \frac{1}{N_i} \sum_{(\mathbf{x}_h, y_h) \in \mathcal{D}_i} CE\big(f(\mathbf{x}_h; w_i), y_h\big), \tag{16}$$

where $CE(\cdot)$ denotes the cross-entropy measure and w_i is initialized by the cluster model w_{C_a}.

For knowledge transfer, client K_i first calls Algorithm 1 and gets the current optimal collaborator set $\mathcal{K}^*$. Then, it requests the models of clients in $\mathcal{K}^*$, and the whole workflow is specified in Sect. 6. Here, we focus on the knowledge distillation details. We

compute the distillation loss as follows:

$$\mathcal{L}^{\text{distill}} = \sum_{(\mathbf{x}_h, y_h) \in \mathcal{D}_i} KL(\mathcal{P}_{i,h}, \mathcal{P}_{\mathcal{K}^*,h}), \tag{17}$$

where $\mathcal{P}_{\mathcal{K}^*}$ denotes the soft labels averaged by teachers [52]. We calculate it similarly to Eq. (15) as follows:

$$\mathcal{P}_{\mathcal{K}^*,h} = softmax\Big(\frac{1}{|\mathcal{K}^*|} \sum_{K_j \in \mathcal{K}^*} \frac{f(\mathbf{x}_h; w_j)}{\tau}\Big). \tag{18}$$

The knowledge distillation is equipped with the two jointly optimized losses as follows:

$$\mathcal{L}^{\text{joint}} = \delta \cdot \mathcal{L}^{\text{predict}} + (1 - \delta) \cdot \mathcal{L}^{\text{distill}}, \tag{19}$$

where δ is a hyperparameter to balance the two losses.

After client K_i finishes the knowledge distillation from teachers, we update the values in $\mathbf{L}_i$ as shown in Eq. (14) for the next selection.

Overall, the objective in FedChain is to minimize the global loss:

$$\mathcal{L}^{\text{global}} = \sum_{C_a \in \mathcal{G}_t} \sum_{K_i \in C_a} \frac{fresh(w_i)}{\sum_{K_j \in C_a} fresh(w_j)} \mathcal{L}^{\text{joint}}(\mathcal{D}_i; w_i). \tag{20}$$

6 THREAT MODEL AND ATTACK DEFENSE

Threat model. We follow the widely-adopted threat model in FL, regarding clients as *honest-but-curious* (a.k.a. *semi-honest*) [40]. In this setting, all clients strictly follow the instructions of FedChain. They do not deviate from the defined procedures but are still curious to learn information about other clients from communication.

With the threat model above and potential attacks, FedChain implements an access control mechanism guided by the following goals: (1) it should ensure secure and authenticated access by verifying the requesters' identities, thereby preventing unauthorized access; (2) it should guarantee the integrity and confidentiality of models for tampering resistance; and (3) it should be efficient and scalable, realizing fault tolerance and load balancing.

Fig. 3 shows the access control mechanism. $enc(\cdot)$, $PriK, PubK$ are the encryption function, private and public keys, respectively:

- **S1.** Client K_i uploads the metadata of local model w_i, consisting of the client identifier ID_i, the identifier of model $ModID_i$, and the hash code $hash(w_i)$.
- **S2–3.** As a requester, client K_j generates a random number X. It sends a message $enc(X, PubK_i)$ to blockchain and a message $\big(ID_j, ModID_i, enc(X, PriK_j)\big)$ to client K_i off-chain.
- **S4–6.** Client K_i decrypts $enc(X, PriK_j)$ and $enc(X, PubK_i)$ using the public key $PubK_j$ and the private key $PriK_i$. K_i verifies the identity of client K_j by comparing the two values of X. Client K_j would be granted access if it is the cluster aggregator or w_i is one of K_j's selected teacher models. Once granted, client K_i sends the model encrypted by $PubK_j$ to client K_j and commits $(ID_j, ModID_i, Granted)$ on-chain.
- **S7.** Client K_i sends the model encrypted by $PubK_j$ to client K_j and commits $(ID_j, ModID_i, Granted)$ on-chain, indicating that K_j keeps a replica of the model.

We introduce a replica mechanism for fault tolerance and load balancing without extra communication costs. In S7, the authorized clients store the model locally as a replica to avoid SPOF. To balance the communication overhead of model transmission, a model requester calculates the replica number by taking the hash code

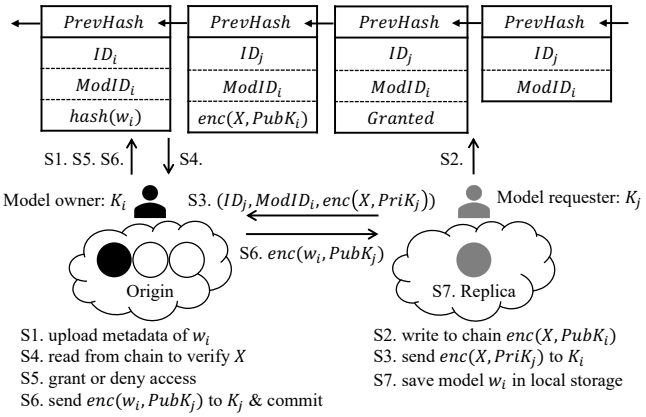

Figure 3: Workflow of the model access request, authorization, and replication

of its own identifier and performing a modulo operation with the total number of available replicas.

Attack defense. We analyze common attacks and explain how the access control mechanism in FedChain defends against them:

- Under an *access control bypass attack* [38], the attacker requests the model directly from the owner without committing to blockchain. As the access control in FedChain is blockchain-based, the bypass requests are ineffective.
- Under a *Sybil attack* [44], the attacker fakes a large number of identities to gain access to the model. But, the attacker cannot replicate multiple blockchain certificates, and any requests from fake identities would not be processed. Thus, a Sybil attack is unable to interfere with the operation.
- Under a *man-in-the-middle attack* [8], the attacker interferes with and manipulates the communications between the model owner and the requester, attempting to steal the model by impersonating the requester. But, the attacker is unable to forge or tamper with the data on-chain. Furthermore, the random number verification presented in S5 can detect a *replay attack*.
- Under a *model tampering attack* [21], as the model owner, the attacker tampers with the local model and sends it to the requester. In FedChain, this attack can be detected when the requester verifies the integrity of the model with the hash code on-chain.

Note that the access control mechanism in FedChain mainly focuses on ensuring the security of model storage and access. Attacks from malicious clients targeting model performance such as model backdoor and data poisoning will be studied in future work.

7 EXPERIMENTS AND RESULTS

7.1 Experiment Preparation

Environment. We develop FedChain in Python 3.7 and deploy it on a server with two Intel Xeon Gold 6326 CPUs, 512GB memory, and four NVIDIA RTX A6000 graphics cards. Hyperledger Fabric [1] is employed as the blockchain network, which runs with the PBFT service for ordering transactions based on endorsements.

Table 1: Federated learning datasets

Datasets	Samples	Classes	Task descriptions
CIFAR-10 [20]	60,000	10	Image classification
CIFAR-100 [20]	60,000	100	Image classification
Shakespeare [30]	417,469	80	Character prediction
Cora [16]	2,708	7	Graph node classification
CiteSeer [16]	3,312	6	Graph node classification

Datasets. Table 1 lists the five benchmark datasets picked. The samples in each dataset are randomly split with the ratio 8:1:1 for training, validation, and testing. We experiment with both IID and non-IID data distributions. Under the IID setting, the whole dataset is randomly divided into local datasets with the same data size for each client. Under the non-IID setting, we use a Dirichlet distribution Dir(β) to simulate the non-IID data distribution between clients, where a smaller β indicates higher data heterogeneity and size deviation. Following [53], we set β to 0.1 and 0.05.

FL competitors. We compare FedChain with two baselines, namely independent and centralized, as well as eight state-of-the-arts, namely FedAvg [30], FedProx [26], FedAsync [47], MOON [25], CSAFL [54], FedAT [7], FedHiSyn [24], and Def-KT [23]. The independent setting means training separately for each client with its local dataset, and the centralized setting means training a global model with the whole dataset by giving up data privacy. Except for Def-KT, other competitors adopt the client-server architecture. Additionally, FedAvg, FedProx, MOON, FedHiSyn, and Def-KT are synchronous. FedAsync is asynchronous, and CSAFL and FedAT are semi-asynchronous. We uniformly call FedAsync, CSAFL, and FedAT asynchronous for simplicity. See Section 2.1 for more details.

Blockchain competitors. We compare FedChain with three blockchain-based FL systems in Section 2.2, namely BlockFL [19], BFLC [27], and ScaleSFL [29], which all use synchronous FL and have open-source implementations. Since the training processes in BlockFL and BFLC are based on FedAvg, we only choose ScaleSFL for comparing the performance in the aspect of FL.

Implementation details. For CIFAR-10 and CIFAR-100, we pick VGG16 [39] as the backbone model. We tune the hyperparameters with grid search. We set the convolution kernel size to 3×3. For Shakespeare, we use randomly initialized embeddings with a dimension of 80 as the input to train a TextCNN model. For Cora and CiteSeer, we adopt GCN with an embedding dimension of 64. We use 10 clients (i.e., nodes in Hyperledger Fabric) in the main experiments. The maximum global round is set to 200. We use early stop based on the average accuracy on the validation sets with a patience of 5 global rounds. The number of local training epochs per round for each client is set to 5, and the learning rate is set to 0.01. For clustering, we set the inter-cluster distance threshold to 0.3. The similarity weight α to adjust the distribution and speed similarity is set to 0.7. The parameter μ for calculating the freshness is set to 0.9. For P2P knowledge transfer, we increase the collaboration probability p_{t_i} from 0.01 to 0.1 with stride 0.01 for every 10 local rounds. The local round threshold λ in collaborator selection is set to 2 and the cost threshold ψ is set to 0.1. The distillation temperature τ is set to 2. The hyperparameter δ to tune the prediction loss and distillation loss is set to 0.6. For access control, we use RSA [5] for asymmetric digital signature and SipHash [2] for hashing. The parameters of the competitors are set according to their papers. We repeat all experiments five times and report the average results.

7.2 Performance in Federated Learning

Accuracy. Table 2 depicts the comparison results of the average accuracy, where accuracy is defined as the proportion of correctly predicted samples over the whole test set. (1) FedChain outperforms all competitors under most settings. Compared with the second-best method MOON, it improves accuracy by 1.20 on average. (2) Data distribution has a significant influence on accuracy. The accuracy of all methods decreases under the non-IID setting. (3) The centralized setting achieves the highest accuracy and the independent setting obtains the lowest. The two baselines represent the upper and lower bounds of accuracy, respectively. (4) The accuracy of asynchronous methods is much worse than that of synchronous methods as they sacrifice accuracy in exchange for high efficiency. As a semi-asynchronous method, FedChain outperforms asynchronous methods and even synchronous ones, showing the effectiveness of clustering and P2P knowledge transfer. (5) FedChain largely outperforms Def-KT, which uses P2P mutual knowledge transfer without model aggregation. The reason is that pure P2P knowledge transfer lacks a consistent gradient optimization direction. (6) FedChain holds a lead of clustered FL, FedHiSyn and CSAFL, because it considers data and device heterogeneity to cluster clients. Moreover, FedChain is capable of re-clustering to avoid local optimum.

Training time. Table 3 lists the comparison results of the average training time to get a converged global model. (1) FedAsync achieves the shortest training time, but its accuracy is far worse than that of many others. (2) On average, FedChain has the third-shortest training time and it is more obvious under the non-IID setting. Faster convergence is one main reason for reducing the training time. Another reason is the parallel aggregation of multiple clusters. (3) The training time of FedChain is not the best among semi-asynchronous methods, but it strikes a balance between efficiency and accuracy. (4) FedChain performs more efficiently than other clustered FL. In addition to mitigating the straggler effect caused by device heterogeneity, FedChain also considers data heterogeneity for clustering, which leads to faster convergence, especially under the non-IID setting.

Convergence curves. Fig. 4 depicts the convergence curves of the training processes based on the average accuracy of the validation sets. We compare with synchronous methods and asynchronous methods under the IID and non-IID settings, respectively. (1) FedChain converges faster than other methods because re-clustering divides clients with similar data features and computing power into the same cluster. (2) FedChain shows less superiority on the IID datasets as each cluster model owns less knowledge than one global model. The non-IID setting causes data imbalance among clients, but FedChain shows larger superiority in speed and accuracy. The main reason is that P2P knowledge transfer enables local models to acquire knowledge from a global scope, which resolves data heterogeneity and client drift. (3) FedChain converges with higher accuracy than asynchronous methods and the convergence speed is often faster than other synchronous methods. This verifies the effectiveness of clustered FL and P2P knowledge transfer.

Table 2: Average accuracy comparison

Accuracy	CIFAR-10			CIFAR-100			Shakespeare			Cora			CiteSeer		
	IID	$\beta = 0.1$	$\beta = 0.05$	IID	$\beta = 0.1$	$\beta = 0.05$	IID	$\beta = 0.1$	$\beta = 0.05$	IID	$\beta = 0.1$	$\beta = 0.05$	IID	$\beta = 0.1$	$\beta = 0.05$
Centralized	90.62	90.62	90.62	67.83	67.83	67.83	61.76	61.76	61.76	82.31	82.31	82.31	76.55	76.55	76.55
Independent	78.56	46.23	40.77	45.61	25.09	20.81	41.21	30.42	25.19	49.27	25.96	23.16	61.78	37.61	23.17
FedAvg	87.52	82.64	77.78	61.23	55.41	50.92	53.55	40.06	34.02	81.37	74.03	64.78	73.15	67.66	61.97
FedProx	87.65	81.67	78.72	61.74	56.97	51.41	52.04	41.14	32.52	81.68	74.48	65.46	73.67	67.84	62.33
MOON	**90.03**	83.91	76.39	62.91	58.85	52.69	52.17	39.29	34.31	82.11	75.29	66.61	74.12	68.47	62.45
FedHiSyn	89.54	83.04	77.85	62.59	57.92	51.27	53.91	40.34	33.01	82.07	74.12	65.29	74.23	67.21	60.95
Def-KT	79.42	71.41	62.13	50.62	47.62	40.29	51.74	36.42	29.03	80.12	65.02	59.61	72.08	64.55	52.05
ScaleSFL	86.37	80.03	74.31	61.52	55.16	47.35	52.68	37.28	29.43	81.91	72.19	63.63	72.32	65.31	57.23
FedAsync	85.04	80.71	71.78	58.14	51.37	41.23	43.81	35.09	28.64	77.39	64.65	61.17	67.83	63.41	51.14
CSAFL	87.06	82.21	76.15	62.43	56.76	52.06	51.49	40.01	31.17	80.52	73.53	64.12	72.59	67.14	59.74
FedAT	87.83	81.25	74.98	61.37	54.87	47.89	52.11	38.57	30.16	81.55	74.29	64.07	73.59	67.33	59.39
FedChain	89.51	**84.28**	**80.41**	**63.24**	**60.33**	**54.26**	**54.15**	**42.31**	**36.61**	**82.23**	**76.65**	**68.73**	**74.55**	**69.35**	**63.34**

The best and second-best scores are marked in **bold** and with underline, respectively, except where otherwise indicated.

Table 3: Average training time comparison

Time (min.)	CIFAR-10			CIFAR-100			Shakespeare			Cora			CiteSeer		
	IID	$\beta = 0.1$	$\beta = 0.05$	IID	$\beta = 0.1$	$\beta = 0.05$	IID	$\beta = 0.1$	$\beta = 0.05$	IID	$\beta = 0.1$	$\beta = 0.05$	IID	$\beta = 0.1$	$\beta = 0.05$
Centralized	25.03	25.03	25.03	30.19	30.19	30.19	41.33	41.33	41.33	3.01	3.01	3.01	3.34	3.34	3.34
Independent	11.32	14.47	17.55	12.94	13.81	15.69	18.91	22.12	25.19	2.04	2.15	2.27	2.12	2.34	2.66
FedAvg	46.44	41.63	42.41	66.84	57.61	48.93	72.53	67.17	63.34	4.75	4.23	4.03	5.12	4.93	4.68
FedProx	45.29	42.37	43.87	67.29	58.25	48.58	71.26	66.15	62.39	4.26	4.25	4.07	5.01	4.71	4.59
MOON	44.18	41.25	41.33	62.17	56.86	47.34	69.32	65.73	64.27	4.33	4.22	4.01	4.89	4.56	4.23
FedHiSyn	47.15	44.12	41.94	64.92	59.02	50.13	70.37	67.59	65.22	4.43	4.34	4.12	5.19	5.07	4.77
Def-KT	50.23	47.89	46.41	65.31	62.09	52.19	75.67	71.23	67.36	4.68	4.51	4.26	5.47	5.24	4.99
ScaleSFL	52.38	45.14	43.46	70.21	60.71	50.47	73.84	69.26	66.11	4.84	4.61	4.18	5.35	5.29	4.86
FedAsync	**37.55**	**32.19**	**30.06**	**44.37**	**38.25**	**34.46**	**58.23**	**55.61**	**50.45**	**3.19**	**2.91**	**2.54**	**3.67**	**3.23**	**3.05**
CSAFL	45.03	40.98	39.21	64.55	55.97	44.96	67.26	64.25	60.19	4.29	4.17	3.89	4.83	4.68	4.54
FedAT	40.12	35.39	36.27	58.95	50.22	42.63	62.34	58.83	55.34	3.73	3.58	3.42	4.57	4.25	4.06
FedChain	47.12	38.37	35.41	65.28	54.11	42.27	68.45	62.74	58.47	4.21	3.99	3.74	5.06	4.41	4.18

7.3 Performance in Blockchain

System throughput and latency. We assess the throughput and latency of different blockchain-based FL systems with the open-source tool Caliper.[1] Throughput refers to the number of processed transactions per second (TPS), and latency refers to the average time from commitment to confirmation for a transaction. We evaluate the core interfaces of blockchain systems for FL with different numbers of clients, which include uploading the updated models and querying the latest global models.

Fig. 5 presents the results of throughput and latency comparison on the CIFAR-10 dataset. (1) Generally, the update uploading interface of FedChain exhibits the highest throughput and second-lowest latency compared to other systems. The main reason is that FedChain writes the updated metadata into blockchain. BlockFL and BFLC upload the complete updated parameters on-chain, resulting in an efficiency bottleneck. ScaleSFL performs better with 25 clients. A possible reason is that ScaleSFL implements a shard-level consensus mechanism, which can improve the efficiency of processing write requests. (2) The query interface of FedChain shows the highest throughput and the lowest latency. On one hand, FedChain minimizes the communication overhead by storing only metadata on-chain. On the other hand, replication-based concurrency control is used in our model access mechanism to enable load balancing. (3) As the number of clients deployed on blockchain increases, the latency of the update uploading interface increases due to the ordering and consensus of transactions by clients. Due to the small size of metadata, the increased latency of update uploading does not cause the bottleneck in FedChain, but uploading parameters directly in BlockFL and BFLC results in linearly increasing latency. The query interface remains low latency because queries do not change the ledger state and the read-only transactions are not submitted for ordering. These observations hold on the other four datasets.

Idle time. We record the idle time of each client at each round. By dividing the data of idle time into different intervals and measuring the proportion of each interval, we compare the efficiency of blockchain-based FL systems. Fig. 6 reports the results. (1) FedChain consistently achieves the lowest proportion on all large intervals ($\geq$ 15 seconds). The main reason is that clients with similar training efficiency are grouped into the same cluster, allowing them to query the latest model in the cluster without waiting for a long time. This

[1] https://hyperledger.github.io/caliper/ (last accessed date: Jan. 11, 2024)

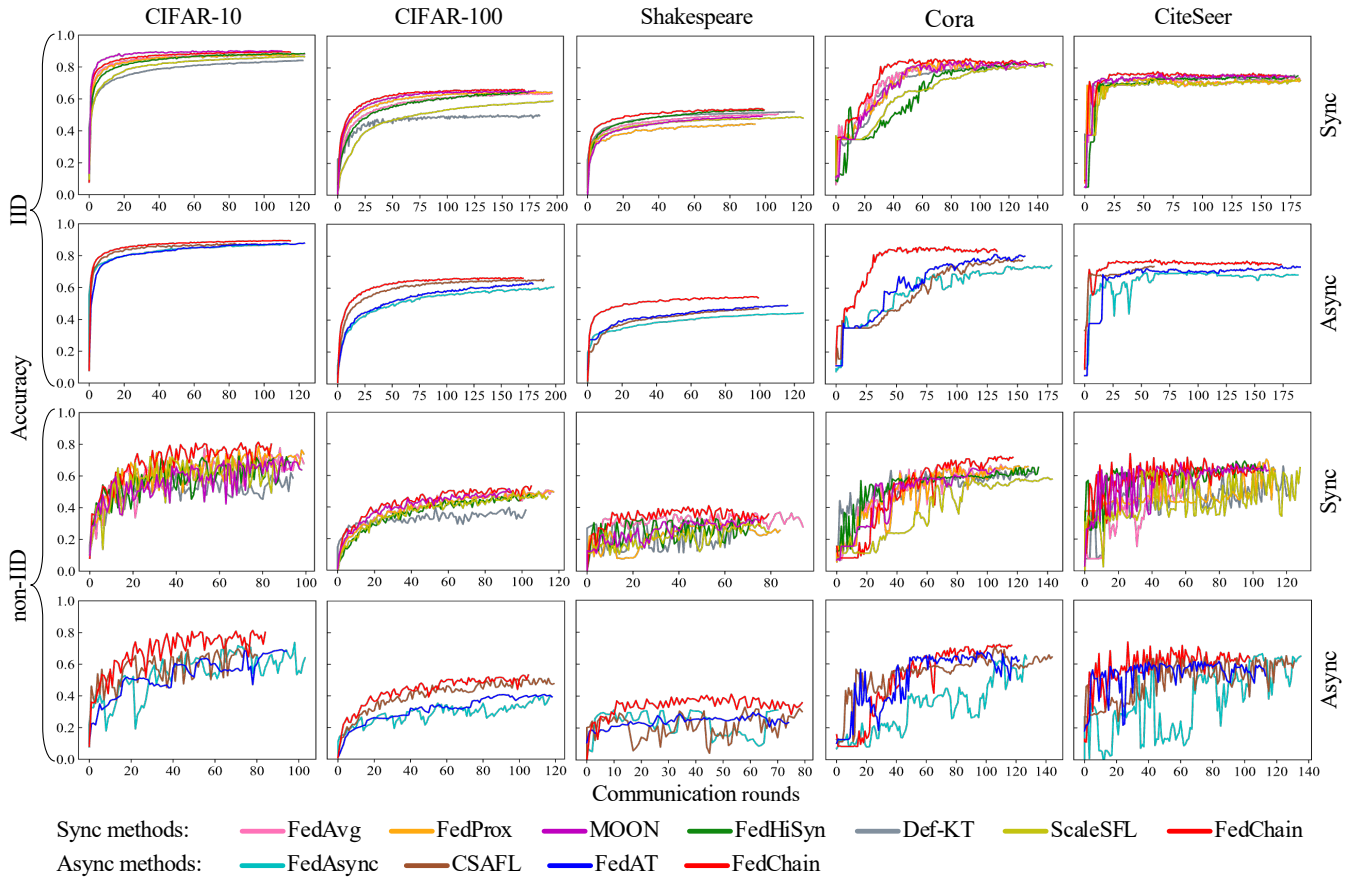

Figure 4: Convergence comparison

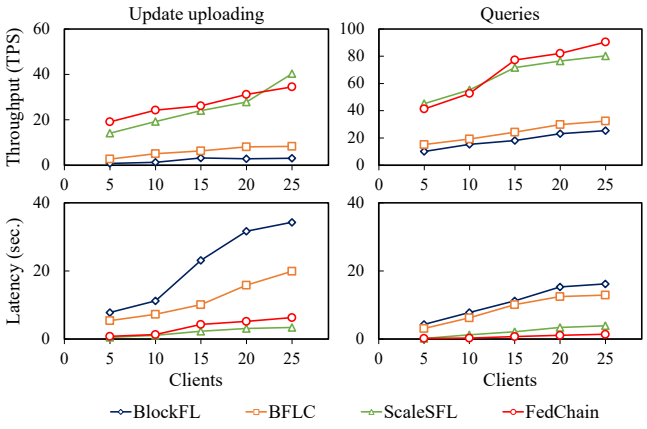

Figure 5: System throughput and latency comparison

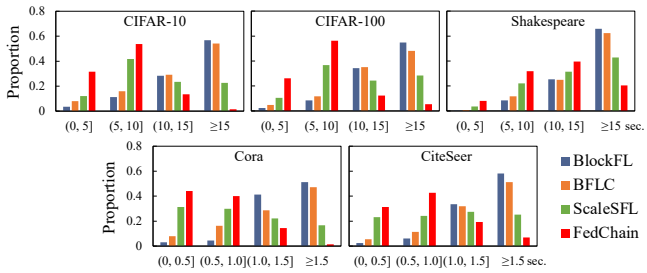

Figure 6: System idle time comparison

shows that FedChain can alleviate the impact of stragglers during the training process. (2) The FL algorithms used by the competing systems are based on FedAvg, and the idle time of clients significantly increases due to waiting for stragglers. This shows that there exists an efficiency bottleneck in synchronous FL.

Impact of blockchain configuration. We conduct an experiment with varying block sizes and transaction arrival rates to evaluate the performance of FedChain. We adjust the configuration of block size with MaxMessageCount. Fig. 7 records the throughput and latency of the update uploading interface on the CIFAR-10 dataset. (1) There is a distinct saturation point at around 200 TPS for the transaction arrival rate. Before reaching the point, the throughput increases linearly with the transaction arrival rate and the latency increases slightly. The throughput decreases and the latency increases significantly after the saturation. The saturation represents the bottleneck of consensus efficiency in FedChain, which

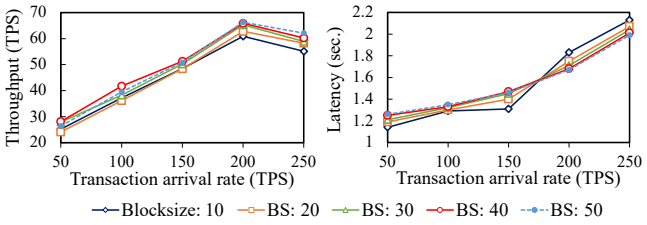

Figure 7: System throughput and latency w.r.t. block size and transaction arrival rate

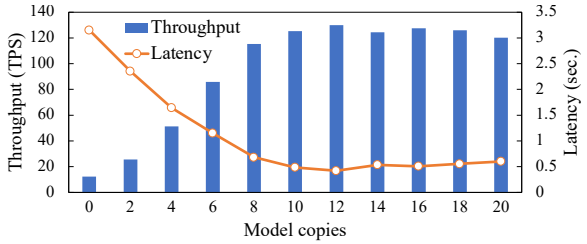

Figure 8: System throughput and latency w.r.t. number of model replicas using 25 clients

mainly comes from the communication cost of consensus process. FedChain leverages the PBFT consensus with a time complexity of $O(|\mathcal{K}|)^2)$. The CPU computing overhead of the consensus process is 110ms on average. Due to the network congestion caused by the increase of transaction arrival rate, the communication cost increases from 137ms to 359ms. This is a possible reason for the apparent increase in latency after the saturation. (2) Before the saturation, increasing the block size causes a slight increase in latency due to a longer block creation time. After the saturation, with the increase of the block size, the latency decreases slightly. This suggests that when consensus efficiency hits the bottleneck, a larger block size can improve throughput and reduce latency. Therefore, at an arrival rate of 200 TPS, increasing the block size from 10 to 50 yields a 12.7% increase in throughput. These observations hold on the other datasets.

Performance w.r.t. number of model replicas. To evaluate the efficiency of access control, we conduct an experiment with 25 clients and varying numbers of replicas for a model to measure the throughput and latency of processing query requests. Fig. 8 presents the results. (1) Increasing the number of model replicas can effectively improve the efficiency of access control. When the number of replicas is less than 12, the throughput increases rapidly and the latency decreases. The main reason is that requests can be distributed across multiple clients with replicas to balance the workload of FedChain. (2) The impact on the efficiency of increasing the number of replicas exhibits marginal utility. When the number of replicas exceeds 8, the growth of the throughput begins to decelerate. Furthermore, the latency increases slightly when the number of replicas exceeds 12. One possible explanation is that the number of requests for model replicas processed in parallel reaches the threshold of transaction processing and ordering on-chain. Therefore, to balance efficiency and replica redundancy, the maximum number of replicas for a model may be set to no more than half of the number of clients.

Table 4: Ablation study under $\beta = 0.05$

Accuracy	CIFAR-10	CIFAR-100	Shakespeare	Cora	CiteSeer
FedChain	**80.41**	**54.26**	**36.61**	**68.73**	**63.34**
w/o clustered FL	78.56	52.71	35.28	67.42	62.28
w/o P2P transfer	79.05	53.74	35.75	67.61	62.17
Time (min.)	**CIFAR-10**	**CIFAR-100**	**Shakespeare**	**Cora**	**CiteSeer**
FedChain	35.41	42.27	58.47	3.74	4.18
w/o clustered FL	41.16	46.58	63.65	4.11	4.75
w/o P2P transfer	**33.67**	**40.05**	**56.29**	**3.46**	**3.84**

7.4 Further Analysis

Ablation study. *First*, we modify FedChain with two variants: (1) FedChain w/o clustered FL, which aggregates all updates into a single global model. (2) FedChain w/o P2P transfer, where clients only perform local training in clustered FL. Table 4 shows the comparison results of average accuracy and training time. (1) Removing either module decreases the performance. The accuracy drops by 1.42 and 1.01 of removing the clustered FL module and the P2P knowledge transfer module, respectively. This indicates that both of them are useful. (2) The training time of FedChain without clustered FL is the highest. The main reason is that certain clients have to wait for the inefficient stragglers to complete their local training. This verifies that the clustered FL plays a key role in improving efficiency. (3) The training time of FedChain is a bit higher than that of FedChain without P2P transfer, but its performance is better, because FedChain selects models from multiple clients as teachers for knowledge distillation instead of pure local training, which requires extra training time.

Second, to assess the impact of the similarity weight α, we compare with FedChain w/o sim_{distr} and FedChain w/o sim_{speed} in Table 5. (1) The accuracy of FedChain without sim_{distr} is lower than the original FedChain, indicating that the effect of clustering is reduced since the similarity of features is not considered. Another reason for the extended training time is that inaccurate clustering leads to slower convergence. (2) The accuracy of FedChain without sim_{speed} drops slightly but the training time increases significantly. This is because clients with similar features may have significant differences in training efficiency and slow down the training speed.

Third, we compare the collaborative client selection policy with the random selection and the top-1 selection policy in Table 6. (1) The accuracy of random selection is lower and the training time is higher than the original FedChain. The reason is that the randomly selected teacher models may not be helpful due to freshness or data heterogeneity, which leads to slower convergence. (2) FedChain with top-1 selection has higher accuracy than random selection but converges at lower accuracy than the original FedChain. The reason is that repeatedly distilling from the most similar client may not lead to sustained knowledge gains.

Hyperparamter setting. *First*, we assess the impact of the freshness threshold λ on overall performance. $\lambda = 0$ means dropping outdated updates directly. Table 7 depicts the results and we observe that FedChain with $\lambda = 2$ performs better than FedChain

Table 5: Analysis of similarity weight α under $\beta = 0.05$

Accuracy	CIFAR-10	CIFAR-100	Shakespeare	Cora	CiteSeer
FedChain	**80.41**	**54.26**	**36.61**	**68.73**	**63.34**
w/o sim_{distr}	78.69	53.08	35.41	67.76	62.07
w/o sim_{speed}	80.22	54.15	36.44	68.42	63.25
Time (min.)	CIFAR-10	CIFAR-100	Shakespeare	Cora	CiteSeer
FedChain	**35.41**	**42.27**	**58.47**	**3.74**	**4.18**
w/o sim_{distr}	38.77	44.32	60.25	3.93	4.53
w/o sim_{speed}	40.82	46.34	62.34	4.08	4.68

Table 6: Analysis of client selection policies under $\beta = 0.05$

Accuracy	CIFAR-10	CIFAR-100	Shakespeare	Cora	CiteSeer
FedChain	**80.41**	**54.26**	**36.61**	**68.73**	**63.34**
w/ random	79.16	53.78	35.31	67.64	62.21
w/ top-1	79.83	53.95	35.92	68.12	62.67
Time (min.)	CIFAR-10	CIFAR-100	Shakespeare	Cora	CiteSeer
FedChain	35.41	42.27	58.47	3.74	4.18
w/ random	37.33	43.59	60.14	4.07	4.39
w/ top-1	**35.11**	**41.72**	**56.98**	**3.61**	**4.05**

with $\lambda = 0$ or $\lambda = 4$, which verifies that recent updates contribute to accuracy improvement and convergence acceleration.

Second, we adjust the cost threshold ψ and show the results in Table 8. (1) The accuracy and efficiency with $\psi = 0.01$ are slightly worse than those with $\psi = 0.1$. This is due to that the number of selected teacher models at each local round is smaller. Clients require more rounds to obtain knowledge from all clients and build a global view. (2) The accuracy with $\psi = 1$ is lower and the training time is higher. Similar to the impact of λ, a large ψ leads to an increasing number of teacher models, which introduces unrelated models as noise and slows down the convergence.

Re-clustering visualization. To show the effectiveness of clustered FL in FedChain, we plot the t-SNE [41] visualization of the data features on CIFAR-10 for five clients. As shown in Fig. 9, we observe that: (1) As the number of rounds increases, both FedChain and ScaleSFL gradually improve their classification abilities based on the data features. FedChain demonstrates more distinct and well-defined categories and boundaries in sample classification. It is because ScaleSFL performs FL based on FedAvg, where a unified global model may struggle to fit all local datasets. In contrast, FedChain achieves a better balance between global convergence and local optimization by clustering clients according to data distributions. (2) Compared to the t-SNE plot of FedChain in 50 rounds, the figure in 100 rounds shows a noticeable shift in client clustering, with increasing distinct boundaries in sample classification. This suggests that the clustered FL in FedChain is capable of adapting dynamically and avoiding local optimum. Furthermore, client clustering and model optimization complement each other. As the models fit more in the local datasets, the clustering accuracy increases. Also, accurate client clustering further results in compatible intra-cluster aggregation towards a converged cluster model.

Table 7: Analysis of freshness threshold λ under $\beta = 0.05$

Accuracy	CIFAR-10	CIFAR-100	Shakespeare	Cora	CiteSeer
$\lambda = 0$	80.19	54.01	36.15	68.38	63.12
$\lambda = 2$	**80.41**	**54.26**	**36.61**	**68.73**	**63.34**
$\lambda = 4$	79.21	52.85	35.14	66.02	62.29
Time (min.)	CIFAR-10	CIFAR-100	Shakespeare	Cora	CiteSeer
$\lambda = 0$	35.97	42.68	58.95	3.85	4.29
$\lambda = 2$	**35.41**	**42.27**	**58.47**	**3.74**	**4.18**
$\lambda = 4$	37.25	45.01	60.23	4.03	4.51

Table 8: Analysis of cost threshold ψ under $\beta = 0.05$

Accuracy	CIFAR-10	CIFAR-100	Shakespeare	Cora	CiteSeer
$\psi = 0.01$	80.24	53.87	36.29	68.45	63.19
$\psi = 0.1$	**80.41**	**54.26**	**36.61**	**68.73**	**63.34**
$\psi = 1$	79.22	53.02	35.34	67.19	62.58
Time (min.)	CIFAR-10	CIFAR-100	Shakespeare	Cora	CiteSeer
$\psi = 0.01$	35.26	43.16	59.63	3.81	4.26
$\psi = 0.1$	**35.41**	**42.27**	**58.47**	**3.74**	**4.18**
$\psi = 1$	39.18	47.33	63.11	4.31	4.69

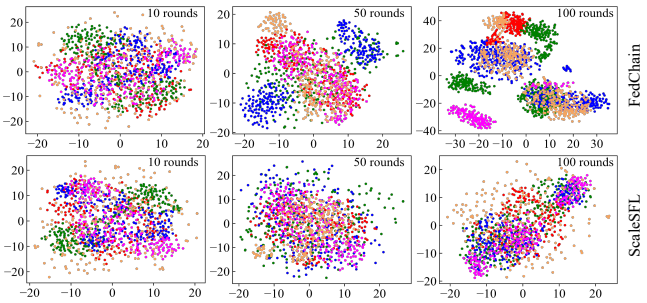

Figure 9: t-SNE plot of features on CIFAR-10, where red, blue, green, pink, and orange dots represent samples from clients 1, 2, 3, 4, and 5, respectively.

8 CONCLUSION

In this paper, we combine blockchain and FL in depth to develop FedChain. We design a clustered semi-asynchronous FL method for model aggregation, mitigating client drift and accelerating training. To optimize local training, we define the NP-hard problem of collaborative client selection and give a multi-client knowledge distillation method based on the P2P network of blockchain. Moreover, we implement an access control mechanism to exchange models safely and efficiently. Experiments on different types of benchmark datasets show that FedChain achieves superior results in accuracy, convergence, throughput, and latency. In future work, we plan to extend FedChain to the IoT scenario, where edge devices are responsible for collecting data while agents carry out model training.

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (No. 62272219).

REFERENCES

[1] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolic, Sharon Weed Cocco, and Jason Yellick. 2018. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In *EuroSys*. ACM, Porto, Portugal, 1–15.

[2] Jean-Philippe Aumasson and Daniel J. Bernstein. 2012. SipHash: A Fast Short-Input PRF. In *INDOCRYPT*. Springer, Kolkata, India, 489–508.

[3] FISCO BCOS. 2017. *Financial Blockchain Open Source Platform FISCO BCOS Whitepaper*. Technical Report. Financial Blockchain Shenzhen Consortium.

[4] Kallista A. Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konečný, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. 2019. Towards Federated Learning at Scale: System Design. In *MLSys*. mlsys.org, Stanford, CA, USA, 374–388.

[5] Dan Boneh. 1999. Twenty Years of Attacks on the RSA Cryptosystem. *Notices of the AMS* 46, 2 (1999), 203–213.

[6] Allan Borodin, Hyun Chul Lee, and Yuli Ye. 2012. Max-Sum diversification, monotone submodular functions and dynamic updates. In *PODS*. ACM, Scottsdale, AZ, USA, 155–166.

[7] Zheng Chai, Yujing Chen, Ali Anwar, Liang Zhao, Yue Cheng, and Huzefa Rangwala. 2021. FedAT: a high-performance and communication-efficient federated learning system with asynchronous tiers. In *SC*. ACM, St. Louis, MO, USA, 1–16.

[8] Mauro Conti, Nicola Dragoni, and Viktor Lesyk. 2016. A Survey of Man In The Middle Attacks. *IEEE Commun. Surv. Tutor.* 18, 3 (2016), 2027–2051.

[9] Hung Dang, Tien Tuan Anh Dinh, Dumitrel Loghin, Ee-Chien Chang, Qian Lin, and Beng Chin Ooi. 2019. Towards Scaling Blockchain Systems via Sharding. In *SIGMOD*. ACM, Amsterdam, The Netherlands, 123–140.

[10] Avishek Ghosh, Jichan Chung, Dong Yin, and Kannan Ramchandran. 2020. An Efficient Framework for Clustered Federated Learning. In *NeurIPS*. Curran Associates, Inc., Online, 19586–19597.

[11] Sreenivas Gollapudi and Aneesh Sharma. 2009. An axiomatic approach for result diversification. In *WWW*. ACM, Madrid, Spain, 381–390.

[12] Bin Gu, An Xu, Zhouyuan Huo, Cheng Deng, and Heng Huang. 2022. Privacy-Preserving Asynchronous Vertical Federated Learning Algorithms for Multiparty Collaborative Learning. *IEEE Trans. Neural Netw. Learn. Syst.* 33, 11 (2022), 6103–6115.

[13] Suyash Gupta, Jelle Hellings, Sajjad Rahnama, and Mohammad Sadoghi. 2020. Building High Throughput Permissioned Blockchain Fabrics: Challenges and Opportunities. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3441–3444.

[14] Mike Hearn. 2016. *Corda: A distributed ledger*. Technical Report. R3.

[15] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the Knowledge in a Neural Network. *CoRR* 1503.02531 (2015), 1–9.

[16] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. In *NeurIPS*. Curran Associates, Inc., Online, 22118–22133.

[17] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J. Reddi, Sebastian U. Stich, and Ananda Theertha Suresh. 2020. SCAFFOLD: Stochastic Controlled Averaging for Federated Learning. In *ICML*. PMLR, Online, 5132–5143.

[18] Leonard Kaufman and Peter J. Rousseeuw. 1990. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, Inc., New York, NY, USA.

[19] Hyesung Kim, Jihong Park, Mehdi Bennis, and Seong-Lyun Kim. 2020. Blockchained On-Device Federated Learning. *IEEE Commun. Lett.* 24, 6 (2020), 1279–1283.

[20] Alex Krizhevsky. 2009. *Learning Multiple Layers of Features from Tiny Images*. Master's thesis. University of Toronto.

[21] Ashish Kundu and Elisa Bertino. 2008. Structural Signatures for Tree Data Structures. *Proceedings of the VLDB Endowment* 1, 1 (2008), 138–150.

[22] Anusha Lalitha, Osman Cihan Kilinc, Tara Javidi, and Farinaz Koushanfar. 2019. Peer-to-peer Federated Learning on Graphs. *CoRR* 1901.11173 (2019), 1–9.

[23] Chengxi Li, Gang Li, and Pramod K. Varshney. 2022. Decentralized Federated Learning via Mutual Knowledge Transfer. *IEEE Internet Things J.* 9, 2 (2022), 1136–1147.

[24] Guanghao Li, Yue Hu, Miao Zhang, Ji Liu, Quanjun Yin, Yong Peng, and Dejing Dou. 2022. FedHiSyn: A Hierarchical Synchronous Federated Learning Framework for Resource and Data Heterogeneity. In *ICPP*. ACM, Bordeaux, France, 1–11.

[25] Qinbin Li, Bingsheng He, and Dawn Song. 2021. Model-Contrastive Federated Learning. In *CVPR*. IEEE, Online, 10713–10722.

[26] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. Federated Optimization in Heterogeneous Networks. In *MLSys*. mlsys.org, Austin, TX, USA, 1–22.

[27] Yuzheng Li, Chuan Chen, Nan Liu, Huawei Huang, Zibin Zheng, and Qiang Yan. 2021. A Blockchain-Based Decentralized Federated Learning Framework with Committee Consensus. *IEEE Netw.* 35, 1 (2021), 234–241.

[28] Bingyan Liu, Yao Guo, and Xiangqun Chen. 2021. PFA: Privacy-preserving Federated Adaptation for Effective Model Personalization. In *WWW*. ACM, Ljubljana, Slovenia, 923–934.

[29] Evan Madill, Ben Nguyen, Carson K Leung, and Sara Rouhani. 2022. ScaleSFL: A Sharding Solution for Blockchain-Based Federated Learning. In *BSCI*. ACM, Nagasaki, Japan, 95–106.

[30] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *AISTATS*. PMLR, Fort Lauderdale, FL, USA, 1273–1282.

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

[32] Dinh C. Nguyen, Quoc-Viet Pham, Pubudu N. Pathirana, Ming Ding, Aruna Seneviratne, Zihuai Lin, Octavia A. Dobre, and Won-Joo Hwang. 2023. Federated Learning for Smart Healthcare: A Survey. *Comput. Surveys* 55, 3 (2023), 60:1–60:37.

[33] Yanqing Peng, Min Du, Feifei Li, Raymond Cheng, and Dawn Song. 2020. FalconDB: Blockchain-based Collaborative Database. In *SIGMOD*. ACM, Portland, OR, USA, 637–652.

[34] Zeshun Peng, Yanfeng Zhang, Qian Xu, Haixu Liu, Yuxiao Gao, Xiaohua Li, and Ge Yu. 2022. NeuChain: A Fast Permissioned Blockchain System with Deterministic Ordering. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2585–2598.

[35] Youyang Qu, Md Palash Uddin, Chenquan Gan, Yong Xiang, Longxiang Gao, and John Yearwood. 2022. Blockchain-enabled Federated Learning: A Survey. *Comput. Surveys* 55, 4 (2022), 1–35.

[36] Abhijit Guha Roy, Shayan Siddiqui, Sebastian Pölsterl, Nassir Navab, and Christian Wachinger. 2019. BrainTorrent: A Peer-to-Peer Environment for Decentralized Federated Learning. *CoRR* 1905.06731 (2019), 1–9.

[37] Felix Sattler, Klaus-Robert Müller, and Wojciech Samek. 2021. Clustered Federated Learning: Model-Agnostic Distributed Multitask Optimization under Privacy Constraints. *IEEE Trans. Neural Netw. Learn. Syst.* 32, 8 (2021), 3710–3722.

[38] Erez Shmueli, Ronen Vaisenberg, Yuval Elovici, and Chanan Glezer. 2010. Database Encryption – An Overview of Contemporary Challenges and Design Considerations. *ACM SIGMOD Record* 38, 3 (2010), 29–34.

[39] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *ICLR*. OpenReview.net, San Diego, CA, USA, 1–14.

[40] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang, and Yi Zhou. 2019. A Hybrid Approach to Privacy-Preserving Federated Learning. In *AISec*. ACM, London, UK, 1–11.

[41] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. *Journal of Machine Learning Research* 9, 11 (2008), 2579–2605.

[42] Ángel Jesús Varela-Vaca and Antonia M Reina Quintero. 2021. Smart Contract Languages: A Multivocal Mapping Study. *Comput. Surveys* 54, 1 (2021), 1–38.

[43] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S Rellermeyer. 2020. A Survey on Distributed Machine Learning. *Comput. Surveys* 53, 2 (2020), 1–33.

[44] Yue Wang, Ke Wang, and Chunyan Miao. 2020. Truth Discovery against Strategic Sybil Attack in Crowdsourcing. In *KDD*. ACM, Virtual, 95–104.

[45] Chenyuan Wu, Mohammad Javad Amiri, Jared Asch, Heena Nagda, Qizhen Zhang, and Boon Thau Loo. 2022. FlexChain: An Elastic Disaggregated Blockchain. *Proceedings of the VLDB Endowment* 16, 1 (2022), 23–36.

[46] Wentai Wu, Ligang He, Weiwei Lin, Rui Mao, Carsten Maple, and Stephen A. Jarvis. 2021. SAFA: A Semi-Asynchronous Protocol for Fast Federated Learning With Low Overhead. *IEEE Trans. Comput.* 70, 5 (2021), 655–668.

[47] Cong Xie, Sanmi Koyejo, and Indranil Gupta. 2020. Asynchronous Federated Optimization. In *International OPT Workshop on Optimization for Machine Learning*. OPT, Online, 1–11.

[48] Cheng Xu, Ce Zhang, Jianliang Xu, and Jian Pei. 2021. SlimChain: Scaling Blockchain Transactions through OffChain Storage and Parallel Processing. *Proceedings of the VLDB Endowment* 14, 11 (2021), 2314–2326.

[49] Qiang Yang. 2019. Federated Recommendation Systems. In *BigData*. IEEE, Los Angeles, CA, USA, 1.

[50] Wensi Yang, Yuhang Zhang, Kejiang Ye, Li Li, and Cheng-Zhong Xu. 2019. FFD: A Federated Learning Based Method for Credit Card Fraud Detection. In *BigData Congress*. Springer, Milan, Italy, 18–32.

[51] Xuefei Yin, Yanming Zhu, and Jiankun Hu. 2022. A Comprehensive Survey of Privacy-preserving Federated Learning: A Taxonomy, Review, and Future Directions. *Comput. Surveys* 54, 6 (2022), 131:1–131:36.

[52] Shan You, Chang Xu, Chao Xu, and Dacheng Tao. 2017. Learning from Multiple Teacher Networks. In *KDD*. ACM, Halifax, NS, Canada, 1285–1294.

[53] Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan H. Greenewald, Trong Nghia Hoang, and Yasaman Khazaeni. 2019. Bayesian Nonparametric

Federated Learning of Neural Networks. In *ICML*. PMLR, Long Beach, CA, USA, 7252–7261.

[54] Yu Zhang, Moming Duan, Duo Liu, Li Li, Ao Ren, Xianzhang Chen, Yujuan Tan, and Chengliang Wang. 2021. CSAFL: A Clustered Semi-Asynchronous Federated Learning Framework. In *IJCNN*. IEEE, Shenzhen, China, 1–10.

[55] Zhebin Zhang, Dajie Dong, Yuhang Ma, Yilong Ying, Dawei Jiang, Ke Chen, Lidan Shou, and Gang Chen. 2021. Refiner: A Reliable Incentive-Driven Federated Learning System Powered by Blockchain. *Proceedings of the VLDB Endowment* 14, 12 (2021), 2659–2662.