FedGraph: Federated Graph Learning With Intelligent Sampling

Fahao Chen, Peng Li, *Senior Member, IEEE*, Toshiaki Miyazaki, *Senior Member, IEEE*, and Celimuge Wu, *Senior Member, IEEE*

Abstract—Federated learning has attracted much research attention due to its privacy protection in distributed machine learning. However, existing work of federated learning mainly focuses on Convolutional Neural Network (CNN), which cannot efficiently handle graph data that are popular in many applications. Graph Convolutional Network (GCN) has been proposed as one of the most promising techniques for graph learning, but its federated setting has been seldom explored. In this article, we propose FedGraph for federated graph learning among multiple computing clients, each of which holds a subgraph. FedGraph provides strong graph learning capability across clients by addressing two unique challenges. First, traditional GCN training needs feature data sharing among clients, leading to risk of privacy leakage. FedGraph solves this issue using a novel cross-client convolution operation. The second challenge is high GCN training overhead incurred by large graph size. We propose an intelligent graph sampling algorithm based on deep reinforcement learning, which can automatically converge to the optimal sampling policies that balance training speed and accuracy. We implement FedGraph based on PyTorch and deploy it on a testbed for performance evaluation. The experimental results of four popular datasets demonstrate that FedGraph significantly outperforms existing work by enabling faster convergence to higher accuracy.

Index Terms—Federated learning, graph learning, graph sampling, reinforcement learning

1 Introduction

Federated learning has shown great promise in enabling collaborative machine learning among distributed devices while preserving their data privacy [1]. There is a growing amount of research efforts on federated learning [2], [3], but they study Convolutional Neural Network (CNN) models that show superior learning accuracy on image and voice data. However, many applications generate graph data (e.g., social graphs and protein structures) consisting of nodes and edges, and much evidence has shown that CNN cannot efficiently handle graph learning [4], [5]. Graph Convolutional Network (GCN) [6] has been proposed to deal with graph learning by a novel graph convolution operation. Different from CNN's convolution operation that filters a small set of neighboring pixels, a graph convolution operation filters the features of neighboring nodes. Unfortunately, existing work of federated learning mainly focuses on CNN, leaving GCN under explored.

Recently, there are several preliminary research efforts about graph learning on decentralized datasets. Zhou *et al.* [7] have studied a vertical federated learning scenario on graphs, where clients maintain the same nodes but with different features and edge types. Similarly, Mei *et al.* [8] assume that graph structural, features and labels belong to different sources. These works are different from the general setting studied in our paper. Some recent works explore the intersection of graph and federated learning by discussing the effect of Non-I.I.D data distribution in federated graph learning [9], [10]. However, these works do not consider the inter-graph connections, which is a pervasive phenomenon in the real world [11].

In this paper, we study federated learning on GCN based on graph data distributed among multiple computing clients that do not allow direct data sharing due to privacy protection. Each client has a subgraph with edge connections to the subgraphs held by others. Every graph node is associated with some features that contain private information. For example, medical records in hospitals can be organized as graphs, where each graph node represents a record and its features include personal information (e.g., ages, genders, and occupations) as well as health conditions (e.g., diseases) [11]. It has been widely recognized that these feature data is privacy-sensitive and they cannot be exposed. Given some nodes with labels, the goal of graph learning is to predict the labels of other nodes.

Federated learning on GCN is not a simple extension of its counterpart on CNN because of two unique challenges. First, GCN training involves node feature sharing among clients, leading to the risk of privacy leakage. To exploit graph structure information, the graph convolution operation is designed to aggregate feature data of neighboring nodes. Such an operation would fail if some neighboring nodes are maintained by other clients, who refuse to expose their features. A straightforward solution for privacy protection is to eliminate feature sharing, but it would seriously

- *Fahao Chen, Peng Li, and Toshiaki Miyazaki are with the University of Aizu, Aizuwakamatsu 965-8580, Japan. E-mail: {d8232101, pengli, miyazaki}@u-aizu.ac.jp.*
- *Celimuge Wu is with the University of Electro-Communications, Chofu 182-8585, Japan. E-mail: celimuge@uec.ac.jp.*

Manuscript received 30 Apr. 2021; revised 24 Sept. 2021; accepted 31 Oct. 2021. Date of publication 8 Nov. 2021; date of current version 9 Dec. 2021.
This work was supported in part by The Okawa Foundation for Information and Telecommunications, in part by G-7 Scholarship Foundation, and in part by JSPS KAKENHI under Grants 21H03424 and 19K20258.
(Corresponding author: Peng Li.)
Recommended for acceptance by J. Zola.
Digital Object Identifier no. 10.1109/TPDS.2021.3125565

decrease training accuracy, which has been confirmed by our experimental results. The second challenge is the high training overhead incurred by large graph size [12], [13]. For example, a social network maintained by Facebook contains over 3 billion users, and the corresponding graph data size may be several hundreds of gigabytes [14]. Since a GCN model stacks several layers of the same structure with the original graph, the model size becomes extremely large, even exceeding the physical memory constraint.

In this paper, we propose FedGraph, a federated graph learning system that integrates the ideas of federated learning and GCN to open new opportunities for privacy-preserving distributed graph learning. FedGraph is especially good at learning on distributed graphs with complicated connections, and can converge to a high training accuracy by addressing the above challenges. For the first challenge about the dilemma of feature sharing and privacy protection, a common solution is to use cryptography-based techniques, e.g., homomorphic encryption [15], [16], to enable computation over encrypted data. Despite strong security guarantee, these techniques have high computational overhead, making them inappropriate choices for FedGraph that pursuits high training speed. There also exist hardware-based solutions, e.g., SGX [17], [18], for privacy protection, but security hardware has limited capacity and it cannot handle large graph data [18]. FedGraph solves the dilemma by designing a cross-client graph convolution operation, without heavy cryptographic operations or dedicated hardware. Instead of directly sharing node features, FedGraph embeds them into low-dimensional representations before sharing, so that original features cannot be recovered.

To reduce GCN training overhead, graph sampling has been widely adopted to randomly select a mini-batch of nodes for training [12], [19], [20], [21]. GraphSAGE [12] is a graph sampling method based on node neighboring relationship. It randomly selects a fixed number of neighbors when applying the graph convolution operation for each node. FastGCN [20] has been proposed to improve sampling efficiency by independently selecting nodes for each graph convolution layer. However, existing work cannot satisfy the requirements of FedGraph design due to three weaknesses. First, these sampling methods depend on some hand-crafted parameters that rely heavily upon the knowledge of domain experts. For example, the performance of GraphSAGE is determined by the parameter specifying the number of sampled neighbors, and manual parameter tuning is time-consuming. Second, existing methods ignore the tradeoff between training speed and training accuracy. Sampling fewer nodes accelerates training but decreases accuracy. Third, clients participating in federated graph learning are heterogeneous in graph size and computational capability. Applying the same sampling policy for all clients is far from the optimal solution.

These weaknesses make the sampling algorithm design challenging in FedGraph. Instead of struggling to improve existing heuristic designs, we resort to Deep Reinforcement Learning (DRL) techniques and design an intelligent sampling algorithm that can automatically adjust sampling policies by jointly considering computation overhead, training accuracy and client heterogeneity. By carefully examining various DRL algorithms, we choose the Deep Deterministic Policy Gradient (DDPG) and cast it to federated graph learning. The main contributions of this paper are as follows.

1) We propose FedGraph as a novel federated graph learning system. We formally present the procedures of local training by clients and global parameter update by the server. A lightweight cross-client convolution operation is proposed to enable feature sharing among clients while avoiding privacy leakage.
2) A DRL-based sampling algorithm is designed for FedGraph, so that it can automatically find the best sampling policy that makes a good tradeoff between training speed and accuracy.
3) We implement a prototype of FedGraph and evaluate it on a testbed. Four popular graph datasets are used in performance evaluation. The experimental results show that FedGraph enables at least 2 times faster convergence to about 10% higher accuracy than existing work.

The rest of this paper is organized as follows. We review some necessary background of GCN and federated learning in Section 2. The FedGraph design is presented in Section 3, followed by the intelligent sampling policy design in Section 4. Section 5 gives experimental results. Related work is presented in Section 6. Finally, Section 7 concludes this paper.

2 Background and Motivation

In this section, we present some necessary backgrounds of federated learning and GCN. In addition, we analyze existing graph sampling approaches as well as their weaknesses, which motivate FedGraph design in this paper.

2.1 Federated Learning

The goal of federated learning is to train a shared model among distributed devices while avoiding the exposure of their training data. A typical federated setting consists of a number of devices, each of which holds a dataset that cannot be exposed to others. In addition, there is a parameter server responsible for synchronizing training results among devices. Federated learning contains multiple training rounds. In each training round, devices first download the latest global model from the parameter server and independently conduct training using their local data. Then, they send updated models or model differences back to the parameter server. After collecting training results from all devices, the parameter server integrates them to create a new global model. During the whole training process, devices share only models and it is almost impossible to infer the training data from these models. Due to the protection for training data, federated learning becomes one of the hottest topics in recent years and many important research efforts have been made to address various challenges [2], [22], [23], [24]. However, they all focus on CNN models, and GCN-oriented federated learning is seldom studied.

2.2 Graph Convolutional Network

CNN has achieved great success in learning on euclidean data, e.g., images and videos. However, a large amount of data in practice are expressed as graphs consisting of nodes and edges, which are also called non-euclidean data. Graph

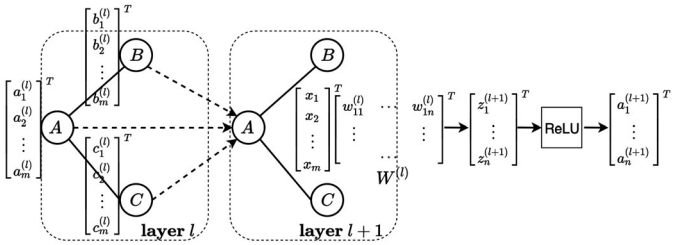

Fig. 1. Illustration of graph convolution operation.

Convolutional Network (GCN) [6] has been proposed as one of the most promising techniques for graph learning. By stacking multiple graph convolutional layers, GCN is able to exploit information of graph structure and node/edge features for node/edge classification problems in various applications. Specifically, we consider an undirected graph defined as $G = (V, E)$, where sets V and E include nodes and edges, respectively. The corresponding graph adjacency matrix is denoted by A. Each node $v \in V$ is associated with a feature vector $x(v)$. A GCN contains L convolutional layers, each of which has the same structure as the original graph G. In the lth layer, each node v is represented by a vector $h^{(l)}(v)$, which is called node embedding. The first layer is the input graph and we have $h^{(1)}(v) = x(v)$. As shown in Fig. 1, the graph convolution operation aggregates embeddings of neighboring nodes, transfers the results into low-dimensional representations, and finally feeds them to an activation function $\sigma(\cdot)$, e.g., ReLU, to generate node embeddings of the next layer. Formally, the propagation rule of GCN can be defined as follows:

$$Z^{(l+1)} = QH^{(l)}W^{(l)}; \quad H^{(l+1)} = \sigma(Z^{(l+1)}), \tag{1}$$

where $H^{(l)}$ includes all node embeddings in the lth layer, and $Q = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$. For the matrix $\tilde{D}$, we have $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ and $\tilde{A} = A + I$, where I is an identity matrix. The feature weights included in $W^{(l)}$ are trainable parameters. Given some nodes with labels, we can train the feature weight matrix $W^{(l)}$ using gradient descent algorithms. The trained parameters can be used to classify the nodes without labels.

2.3 Graph Sampling

In many applications, graphs are very large and the corresponding GCN training has high computational overhead. Graph sampling has been proposed to reduce the sizes of graphs used for GCN training, and its existing work can be classified into two categories. One is node-wise neighbor-sampling that iteratively samples a fixed number of neighbors

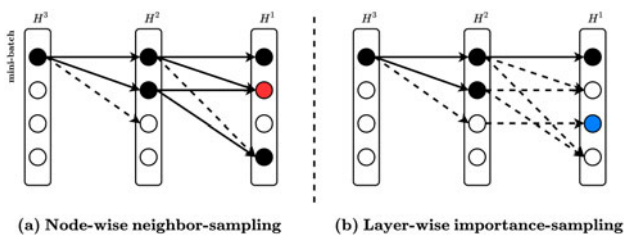

Fig. 2. An illustration of different sampling approaches. The sampled nodes are marked in color (dark, red, and blue). The dashed arrows denote edge connections in the original graph. The solid arrows denote the edges preserved by sampled nodes.

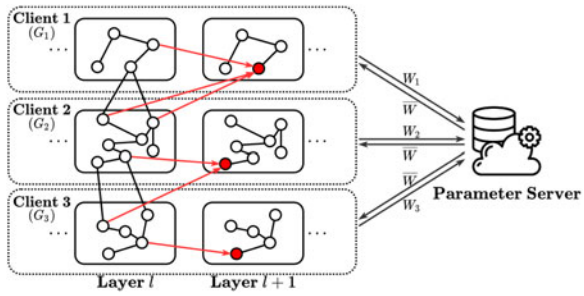

Fig. 3. The FedGraph architecture. Each client i maintains a local graph G_i. During the training, nodes in the mini-batch (nodes in red) aggregate neighbors' embeddings to generate the next layer's embeddings, denoted by red arrows. When training completes, each client i uploads its local model weights W_i to the parameter server. Finally, the parameter server aggregates all local model weights to the updated global model $\bar{W}$ and sends it back to all clients.

for each node. As shown in Fig. 2a, given some nodes in the $(l+1)$th layer, we randomly select a subset of their neighbors as the lth layer. Such a sampling guarantees that aggregation of node embeddings always happens among neighboring nodes. A representative work of node-wise neighbor-sampling is GraphSAGE [12]. However, the number of sampled nodes may exponentially increase as more layers are constructed. In addition, Huang *et al.* [25] have pointed out that it incurs redundancy of embedding calculation at some nodes, e.g., the red nodes in Fig. 2a, which are the shared neighbors of other nodes. Several recent approaches, e.g., VR-GCN [19] and Cluster-GCN [26], have been proposed to improve the performance of node-wise neighbor-sampling, but they cannot fundamentally address this weakness.

The other kind of approaches is called layer-wise importance-sampling. Its basic idea is to independently sample a fixed number of nodes for each GCN layer based on a sampling probability, which is calculated based on node degrees. FastGCN [20] is a typical approach of layer-wise importance-sampling. However, since nodes of different layers are sampled independently, some sampled nodes may have no connections with the ones in the previous layer, like the blue-marked node shown in Fig. 2b. The embeddings of some unlinked nodes may be lost during graph convolution operations, which would deteriorate the training performance.

The strengths and weaknesses of both sampling approaches motivate us to design a new sampling policy that can well control the computation overhead while keeping neighboring relations during sampling.

3 FedGraph Design

We consider a typical setting of federated graph learning, which consists of a set C of computing clients that conduct local training tasks, and a server responsible for global parameter update, as shown in Fig. 3. Computing clients and the server may locate at different locations and they are connected by wide-area networks. Each client $i \in C$ maintains a graph $G_i(V_i, E_i)$, where each node $v \in V_i$ is associated with a feature vector $x(v)$ that cannot be exposed to other clients. A subset $V_i^{label} \subseteq V_i$ of nodes have labels denoted by $\{y(v)|v \in V_i^{label}\}$, which can be used as training data. The edge set E_i contains the internal edges among nodes in V_i, as well as the external ones connecting to nodes held by

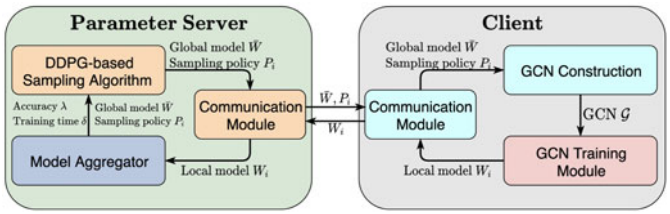

Fig. 4. System design.

other clients. Each client is aware of the existence of neighboring nodes maintained by others but cannot directly access their feature vectors.

We assume that computing clients and the parameter server are *honest-but-curious*, i.e., they honestly follow the federated learning procedures but want to learn feature information held by others. This is a typical threat model that has been widely used by current federated learning research [15], [27], [28]. Some other more serious threat models are discussed as follows. Some malicious clients can tamper with the training by modifying model parameters sent to the parameter server. To deal with this threat, we can use Trusted Execution Environment (TEE) for local training. TEE is commonly available on modern CPUs. It enables an isolated execution environment guaranteed by hardware, and adversaries cannot access data and codes in TEE. Besides, malicious parameter servers can modify global model parameters to compromise federated learning. We can use secure multi-party computation (MPC) or homomorphic encryption (HE) to protect the model aggregation. Besides, TEE can be also used to protect global model aggregation at the parameter server.

Our system design is shown in Fig. 4. We customize the parameter server and clients by adding new modules to implement intelligent sampling. The parameter server contains three main modules. The DDPG-based sampling algorithm generates sampling policies for all clients. A model aggregator collects local feature weights trained by clients and aggregates them to generate new global feature weights for next-round training. In addition, a communication module is designed for message exchanges between the parameter server and clients. This communication module is realized by gRPC APIs, which are based on TCP communication protocol. Each client has a module of GCN construction, responsible for creating a GCN model according to sampling policy. A GCN training module is designed to run the training algorithm.

In FedGraph, in order to predict $y(v)$ of unlabeled nodes, clients collaboratively train global feature weights $\overline{W}$. There are multiple training rounds. In each round, clients download the latest feature weights from the server and construct local GCNs to train these weights. Due to the existence of external edge connections, local GCN training involves embedding sharing among clients. After that, they send updated feature weights to the server, which then creates new global feature weights that will be used for the next-round training. Although FedGraph shares a similar process with traditional federated learning, it has unique procedures of local training and global parameter update, which are presented as follows.

3.1 Local GCN Training by Clients

The local GCN training procedure of each client $i \in C$ is described in the Algorithm 1. At the beginning of each round, client i downloads the latest feature weights $\overline{W}$ as well as a graph sampling policy P_i from the server. The local feature weights are initialized as $W_i = \overline{W}$. Then, this client launches multiple training iterations to update feature weights based on local graph data. Specifically, each training iteration consists of the following two main steps.

Algorithm 1. Local Training Procedure of Client $i \in C$

for each training round t **do**
  Download the latest feature weights $\overline{W}$ and a sampling policy P_i from the parameter server;
  Initialize the local feature weights as $W_i = \overline{W}$;
  for each iteration **do**
    Construct a GCN $\mathcal{G}_i = \{\mathcal{V}_i^{(1)}, \mathcal{V}_i^{(2)}, \ldots, \mathcal{V}_i^{(L)}\} =$ **ModelConstruct**(G_i, P_i)
    for each layer $l = 1, 2, \ldots, L-1$ **do**
      for each node $v \in \mathcal{V}_i^{(l)}$ **do**
        if $l = 1$ **then**

$$z_i^{(l+1)}(v) = \sum_{u \in V_i} \widetilde{Q}_i^{(l)}(v,u) h_i^{(l)}(u) W_i^{(l)}; \tag{2}$$

        else if $l > 1$ **then**

$$z_i^{(l+1)}(v) = \sum_{u \in V_i} \widetilde{Q}_i^{(l)}(v,u) h_i^{(l)}(u) W_i^{(l)} + \sum_{j \in C}^{j \neq i} \sum_{u \in V_j} \widetilde{Q}_i^{(l)}(v,u) h_j^{(l)}(u) W_j^{(l)}; \tag{3}$$

        end if
        Generate the embeddings of the $(l+1) - th$ layer:

$$h_i^{(l+1)}(v) = \sigma(z_i^{(l+1)}(v)); \tag{4}$$

      end for
    end for
    Calculate the loss according to the function:

$$\mathcal{L} = \frac{1}{|\mathcal{V}_i^{(L)}|} \sum_{v \in \mathcal{V}_i^{(L)}} loss(y(v), z_i^L(v)) \tag{5}$$

    Update the local feature weight:

$$W_i \leftarrow W_i - \epsilon \nabla \mathcal{L} \tag{6}$$

  end for
  Submit updated feature weights W_i to the server;
end for

3.1.1 GCN Construction

We construct a GCN $\mathcal{G}_i$ of L layers, using the function *ModelConstruct*() that samples a subset of nodes according to the policy P_i. The basic idea is to start by randomly selecting a set of nodes with labels, which is also referred to as a mini-batch. For each node in the mini-batch, we then iteratively aggregate the embeddings of a sampled subset of neighbors at most $L-1$ hops away.

The pseudo codes of *ModelConstrut*() are shown in Algorithm 2. Specifically, a sampling policy can be expressed by $P_i = \{\kappa_i, p_i^{(1)}, p_i^{(2)}, \ldots, p_i^{(L-1)}\}$, where κ_i denotes the mini-batch size, and $\{p_i^{(1)}, p_i^{(2)}, \ldots, p_i^{(L-1)}\}$ are neighbor sampling probabilities of $L-1$ layers, respectively. As shown in line 1, we sample κ_i labeled nodes as the mini-batch and they compose the final Lth layer. Then, we iteratively construct other GCN layers in a backward direction. For each node v in the $(l+1)$th layer, we randomly select a subset $N_i^{(l)}(v)$ of its neighbors into the lth layer with a probability $p_i^{(l)}$. In addition, we create a matrix $\widetilde{Q}_i^{(l)}$ to replace Q in (7), where $V_i(v)$ denotes the set of neighbors of node v in the original graph G_i. The matrix $\widetilde{Q}_i^{(l)}$ describes the updated adjacent relation after sampling, and it will be used for feature aggregation later. All sampled nodes in the lth layer are maintained in set $\mathcal{V}_i^{(l)}$, as shown in the final line.

Algorithm 2. The Pseudocodes of **ModelConstruct**()

1: Randomly select κ_i labeled nodes as a mini-batch and include them into the Lth layer, i.e., $\mathcal{V}_i^{(L)}$;
2: **for** each layer $l = L-1, \ldots, 2, 1$ **do**
3: **for** each node $v \in \mathcal{V}_i^{(l+1)}$ **do**
4: Sample a subset $N_i^{(l)}(v)$ of neighbors according to a selection probability $p_i^{(l)}$;
5: Update the adjacent matrix $\widetilde{Q}_i^{(l)}$ as follows.

$$\widetilde{Q}_i^{(l)}(v,u) = \begin{cases} \frac{|V_i(v)|}{|N_i^{(l)}(v)|} Q_i(v,u), & \text{if } u \in N_i^{(l)}(v); \\ 0, & \text{otherwise}; \end{cases} \tag{7}$$

6: **end for**
7: $\mathcal{V}_i^{(l)} = \cup_{v \in \mathcal{V}_i^{(l+1)}} N_i^{(l)}(v)$;
8: **end for**

The GCN construction combines the strength of node-wise sampling and layer-wise sampling. These sampling probabilities are independent, which offers opportunities for fine-grained sampling over layers, like layer-wise sampling. By carefully setting these probabilities, we can avoid the high computational cost incurred by the recursive explosive expansion of the neighborhood. Meanwhile, since the sampling process is based on neighborhood relation, which is similar to node-wise sampling, we can avoid sampling nodes without connections.

3.1.2 GCN Training

After constructing a GCN model, we continue to train this GCN based on gradient descent. The cross-client graph convolution operation is described in lines 7-13 of Algorithm 1. Specifically, clients aggregate embeddings of only internal neighbors when they process the first GCN layer, as shown in Eq. (2). From the second layer, we enable clients to aggregate both internal neighbors and external ones, which is shown in Eq. (3). Such a design can prevent the leakage of local origin features while enabling information sharing. We will give the security analysis in Section 3.3.

After aggregation, a nonlinear transformation is applied to generate the node embedding $h_i^{(l+1)}(v)$ of the next layer, as shown in Eq. (4). With the objective of minimizing a loss function defined in Eq. (5), we compute the gradients and update feature weights in Eq. (6), where ϵ is the learning rate. Finally, client i submits the updated feature weights (or their differences from downloaded ones) to the parameter server.

3.2 Global Parameter Update by the Server

The procedure of global weight update by the parameter server is shown in Algorithm 3. The server starts by initializing random feature weights $\overline{W}$ and sampling policies $\{P_1, P_2, \ldots P_{|C|}\}$, and then sends them to clients, respectively. In each of the following training rounds, it collects updated local feature weights from all clients, followed by two main tasks. First, it creates global feature weights by aggregating local weights as shown in Eq. (8), where κ_i denotes the mini-batch size, i.e., the number of labeled nodes, at client i in the current training round. The second task is to update sampling policies for clients using function *GenSampling()*, whose details will be given in the next section. The design of *GenSampling()* is one of the most important contributions of this paper, and it relies on the deep reinforcement learning technique to balance computational overhead and model accuracy. Finally, the server sends new global feature weights and sampling policies to clients to start the next round of training.

Algorithm 3. Global Weight Update of Parameter Server

1: Initialize random feature weights $\overline{W}$ and sampling policies $\{P_1, P_2, \ldots P_{|C|}\}$, and send them to clients, respectively;
2: **for** each training round t **do**
3: Collect feature weights $\{\overline{W}, W_1, W_2, .., W_{|C|}\}$, from all clients;
4: Create global feature weights:

$$\overline{W} = \sum_{i \in C} \frac{\kappa_i}{\sum_{i \in C} \kappa_i} W_i; \tag{8}$$

5: Update the sampling policy $\{P_1, P_2, \ldots, P_{|C|}\} = \mathbf{GenSampling}(\overline{W}, W_1, W_2 \ldots, W_{|C|})$;
6: Send global feature weights $\overline{W}$ and sampling policy P_i to every client $i \in C$;
7: **end for**

3.3 Security Analysis

To show how our proposed Algorithm 1 protects feature data, we consider two clients i and j, who need to share node embeddings during training, without loss of generality. Suppose client i aggregates embeddings from client j and wants to infer the original node features $h_j^{(1)}$. Note that $h_j^{(1)}$ is a matrix containing features of all nodes held by client j, i.e., $h_j^{(1)}(v) = x_j(v), v \in V_j$.

We let V_j^i denote the client i's neighboring nodes at client j. According to Algorithm 1, client i can get information of $\{h_j^{(2)}(V_j^i)W_j^{(2)}, h_j^{(3)}(V_j^i)W_j^{(3)}, \ldots, h_j^{(L)}(V_j^i)W_j^{(L)}\}$. Then, client i can guess node embeddings $\{h_j^{(2)}, \ldots, h_j^{(L)}\}$ by approximating remote $W_j^{(l)}$ using local $W_i^{(l)}$, which is possible when they just synchronize global feature weights from the server.

However, it would be difficult for client i to further infer $h_j^{(1)}(V_j^i)$ because $h_j^{(2)} = \sigma(\tilde{Q}_j^{(1)} h_j^{(1)} W_j^{(1)})$ and client i has no information about $\tilde{Q}_j^{(1)}$, i.e., the adjacent matrix in client j after sampling. Furthermore, the guess of $\{h_j^{(2)}, \ldots, h_j^{(L)}\}$ can hardly achieve high accuracy due to the dimension reduction

of embeddings in higher layers. Given that original features of neighboring nodes can be protected, it would be impossible to get the features of internal nodes at client j. Therefore, we can conclude that FedGraph can protect the node features while enabling information sharing during federated graph learning.

4 Intelligent Graph Sampling Based on DRL

Sampling policies $\{P_1, P_2, \dots P_{|C|}\}$ determine how many nodes are involved in GCN training, and they affect both computational overhead and training accuracy. By sampling fewer nodes, we can accelerate the training process with reduced computational overhead, while lowering training accuracy. On the other hand, with more sampled nodes, we can better approximate the original GCN to achieve higher training accuracy, but incurs a high computational cost. Therefore, it is significant to design sampling policies to make a tradeoff, however, which has been ignored by existing work. Meanwhile, sampling policy design is difficult due to a large optimization space, and manual tuning hardly works in practice. We desire automatic algorithms, with minimum human involvement, to generate good sampling policies.

By carefully examining sampling policies, we find that their influence on the learning performance, in terms of training speed and accuracy, cannot be described using precise closed-form expressions. Instead of struggling with heuristic algorithm design, we resort to Deep Reinforcement Learning (DRL) that can automatically approximate a good solution. The idea of DRL can be implemented in various ways, generating a thriving family of algorithms for different application scenarios with different performance. By carefully comparing candidate DRL algorithms, we choose to use Deep Deterministic Policy Gradient (DDPG) algorithm [29], which can efficiently handle the high-dimensional and continuous action space of our problem. DDPG combines Deep Q-Networks and actor-critic approach and thus enjoys their benefits.

4.1 DDPG-Based Problem Formulation

To apply DDPG, we first formulate our problem as a Markov decision process as follows.

State Space. We define the system state of the training round t as the observed feature weights at the beginning of this round, which can be represented by $s[t] = \{\overline{W}[t], W_1[t], W_2[t], \dots, W_{|C|}[t]\}$. Note that $\overline{W}[t]$ is the global feature weights and $W_i[t]$ denotes the local feature weights of client $i \in C$. The whole action space is denoted by $\mathcal{S}$. Since the state space is huge, we leverage the principal component analysis (PCA) [30] to project the high-dimensional space onto a lower-dimensional space while keeping the distribution information as complete as possible.

Action Space. At the beginning of round t, the parameter server needs to decide graph sampling policies for all clients. The action $a[t]$ of each round t is therefore defined as the corresponding sampling policies, i.e., $a[t] = \{P_1[t], P_2[t], \dots, P_{|C|}[t]\}$. The action space is denoted by $\mathcal{A}$.

Reward. Since both learning speed and accuracy are considered as performance metrics, the reward should be defined to reflect them. We use the completion time of each training round t, which is denoted by $\delta[t]$, to evaluate the training speed. The server can easily obtain $\delta[t]$ by measuring the time consumption of collecting local training results from all clients. The training accuracy $\lambda[t]$ is calculated based on a testing set at the parameter server. We consider a typical federated setting, where the parameter server is usually the task publisher that holds a testing set. Each client has its own training set and validation set, which cannot be exposed due to privacy concerns. With the information of $\delta[t]$ and $\lambda[t]$, we define the reward of each round t as follows,

$$r[t] = \Omega^{(\lambda[t]-\Lambda)} - \alpha(\delta[t] - \beta), \tag{9}$$

where Λ is the target accuracy. The constants Ω, α and β can be adjusted to express different preferences on learning speed and accuracy. The reward contains two parts. The first part evaluates accuracy improvement. We notice that $\lambda[t]$ shows nonlinear improvements as the learning proceeds. It can be quickly improved in the first few training rounds, but the improvement becomes smaller later. In order to make the reward unbiased, we use an exponential function here. The second part evaluates the completion time of each training round in the negative form, to encourage fast training. In practice, the completion time of a client is affected by many factors, i.e., computational hardware or network latency. We alleviate the impact of these factors by adding a constant β in (9), so that we can better evaluate the influence of different sampling policies. In our experiments, we control the time penalty, i.e., $\alpha(\delta[t] - \beta)$, close to 1, as referred to [24], which can be easily achieved by profiling.

Learning Policy and Objective. We define the DRL learning policy in our problem as $\pi_{\boldsymbol{\theta}} : \mathcal{S} \rightarrow \mathcal{A}$, which is parameterized by θ. More precisely, given a state $s[t]$, the algorithm outputs a deterministic action a_t. The objective of our DRL-based sampling algorithm is to maximize the expected cumulative discounted reward from the starting state, which is defined as

$$J(\theta) = \mathbb{E}[R[t]|S[t] = s[t]],$$

where $R[t] = \sum_{k=0}^{\infty} \gamma^k r[t+k]$ is the cumulative discounted reward function.

The action-value function $q_\pi(s[t], a[t])$ is defined to describe the expected cumulative discounted reward after executing action $a[t]$ in state $s[t]$ based on policy π_θ, i.e., $q_\pi(s[t], a[t]) = \mathbb{E}[R[t]|S[t] = s[t], A[t] = \pi_\theta(s[t])]$. Typically, we use neural networks to approximate the policy function π_θ and action-value function q_π.

4.2 Sampling Based on DDPG

The DDPG-based sampling algorithm design is illustrated in Fig. 5. We design an actor network $\mu(s|\boldsymbol{\theta}_\mu)$ to predict deterministic actions, and a critic network $q(s, a|\boldsymbol{\theta}_q)$ to estimate the action-value function $q_\pi(s, a)$. Meanwhile, we maintain copies of the actor network and critic network, denoted by $\tilde{\mu}(s|\tilde{\boldsymbol{\theta}}_\mu)$ and $\tilde{q}(s, a|\tilde{\boldsymbol{\theta}}_q)$, which are also referred to as target networks. They can be used to update the original actor and critic networks.

Similar to Deep Q-Networks, we maintain a replay buffer of finite size to store historical transitions defined as $(s[t], a[t], r[t], s[t+1])$. We update the actor and critic networks by sampling a mini-batch of transitions from the reply buffer. When the buffer is full, the oldest samples are discarded. We then formally introduce the DRL-based

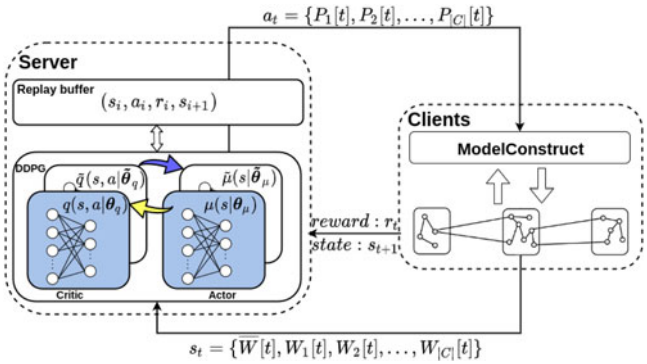

Fig. 5. Illustration of DDPG-based sampling.

sampling algorithm, i.e., implementation details of function *GenSampling*(), and explain how it learns the optimal sampling scheme.

Algorithm 4. Sampling Algorithm Based on DRL

1: Randomly initialize the actor $\mu(s|\boldsymbol{\theta}_\mu)$ and critic $q(s, a|\boldsymbol{\theta}_q)$ with parameters $\boldsymbol{\theta}_\mu$ and $\boldsymbol{\theta}_q$;
2: Initialize the target networks $\tilde{\mu}(s|\tilde{\boldsymbol{\theta}}_\mu)$ and $\tilde{q}(s, a|\tilde{\boldsymbol{\theta}}_q)$ with parameters $\tilde{\theta}_\mu \leftarrow \boldsymbol{\theta}_\mu$ and $\tilde{\theta}_q \leftarrow \boldsymbol{\theta}_q$;
3: Initialize the initial state $s[0] = \{\overline{W}[0], W_1[0], \ldots, W_{|C|}[0]\}$;
4: Reduce the dimension of initial state: $s'[0] = \mathbf{PCA}(s[0])$;
5: Initialize the exploration noise Δ and replay buffer;
6: Generate sampling policies represented by $a[0] = \mu(s'[0]|\boldsymbol{\theta}_\mu) + \Delta_0$ and send them to clients;
7: **for** episode = $1, 2, \ldots, Z$ **do**
8: **for** $t = 1, 2, \ldots, T$ **do**
9: Observer the state $s[t]$ and reward $r[t-1]$;
10: $s'[t] = \mathbf{PCA}(s[t])$;
11: Store the transition $(s'[t-1], a[t-1], r[t-1], s'[t])$ into the replay buffer;
12: Randomly select a mini-batch of K transitions from the replay buffer;
13: Update the critic and actor networks by (11) and (12);
14: Update the target networks by soft update method:

$$\tilde{\boldsymbol{\theta}}_\mu = \phi\boldsymbol{\theta}_\mu + (1-\phi)\tilde{\boldsymbol{\theta}}_\mu, \tag{13}$$

$$\tilde{\boldsymbol{\theta}}_q = \phi\boldsymbol{\theta}_q + (1-\phi)\tilde{\boldsymbol{\theta}}_q; \tag{14}$$

15: Generate sampling policies $a[t] = \mu(s'[t]|\boldsymbol{\theta}_\mu) + \Delta_t$;
16: **end for**
17: **end for**

The pseudo codes of the DDPG-based algorithm are shown in Algorithm 4. We initialize four networks as well as the system state in lines 1–5. At the beginning of training round t, the server observes the current state information $s[t]$ in the form of feature weights of all clients, and the reward $r[t-1]$ defined in (9), as shown in line 9. Then, we reduce the dimension of $s[t]$ to get $s'[t]$ using the PCA method [30], and then store the transition $(s'[t-1], a[t-1], r[t-1], s'[t])$ into the replay buffer. After that, we randomly select a mini-batch of K transitions to update the critic network by minimizing the loss function

$$\mathcal{L} = \frac{1}{K}\sum_{k=1}^{K}(q^{target} - q(s'[t_k-1], a[t_k-1]|\boldsymbol{\theta}_q))^2, \tag{10}$$

TABLE 1
Graph Data Statistics

Dataset	Nodes	Edges	Features	Classes
Cora	2,708	10,556	1,433	7
Citeseer	3,327	9,228	3,703	6
PubMed	19,717	88,651	500	3
Reddit	232,965	114,848,857	602	41

where $q^{target} = r[t_k-1] + \gamma\tilde{q}(s'[t_k], \tilde{\mu}(s'[t_k]|\tilde{\boldsymbol{\theta}}_\mu)|\tilde{\boldsymbol{\theta}}_q)$ is the target action value. The parameters of the critic network are updated by

$$\boldsymbol{\theta}_q[t] = \boldsymbol{\theta}_q[t-1] - \eta_q\nabla\mathcal{L}, \tag{11}$$

where η_q is the learning rate. Then we update the actor network as follows:

$$\boldsymbol{\theta}_\mu[t] = \boldsymbol{\theta}_\mu[t-1] - \eta_\mu\left[\frac{1}{K}\sum_i \nabla_a q(s[i], a)\nabla_{\boldsymbol{\theta}_\mu}\mu(s[i])|_{a=\mu(s[i])}\right], \tag{12}$$

where η_μ is the learning rate of the actor network. The parameters of two target networks are updated in line 14, where $\phi \ll 1$. Finally, we obtain the action $a[t]$ representing sampling policies based on updated networks.

5 Performance Evaluation

5.1 Experimental Settings

We implement FedGraph using PyTorch and Deep Graph Library (DGL) [31], a Python package dedicated to deep learning on graphs. We deploy FedGraph on 20 computing clients with Intel i7-10700 CPU, 32 GB memory, and Geforce RTX 2080 GPU. We consider 4 popular graph datasets: Cora, Citeseer, PubMed, and Reddit, which have been widely used for GCN studies [12], [19], [20], [21], [25], [26]. Some statistic information of these datasets is summarized in Table 1. Since some graphs, e.g., Cora and Citeseer, are with limited sizes, we synthesize large graphs based on these datasets using the following method. Given a dataset in Table 1, each client i randomly selects a proportion ξ_i of nodes as its local graph data, and $\{\xi_1, \xi_2, \ldots, \xi_{|C|}\}$ belongs to a normal distribution with a mean of 0.8. It is possible that generated local graphs overlap on some nodes, especially for small graph datasets, like Cora and Citeseer. For large graphs, we carefully control the local graph generation to avoid overlapping. Even some nodes overlap in the synthesized datasets, we treat them as different nodes and there is no influence to training performance. A similar graph synthesis method has been adopted by [32]. For the local dataset, we randomly choose a set of nodes to generate a training set, a validation set, and a test set. The edge connections across clients are maintained according to the original graph. For local graph learning, each client constructs a 3-layer GCN, including an input layer and two convolutional layers. We set 16 hidden units, 50% dropout rate, 0.01 learning rate for Cora, Citeseer, and PubMed. For Reddit, there are 128 hidden units, the dropout rate is 20%, and the learning rate is 0.0001. We set the batch size as 256 for Cora, Citeseer, and Reddit, 1,024 for PubMed [20]. We use ADAM optimizer for local GCN training. For

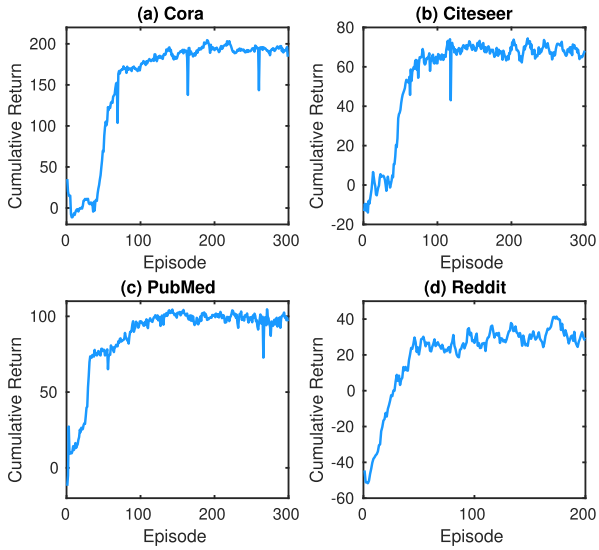

Fig. 6. Cumulative discounted returns of FedGraph.

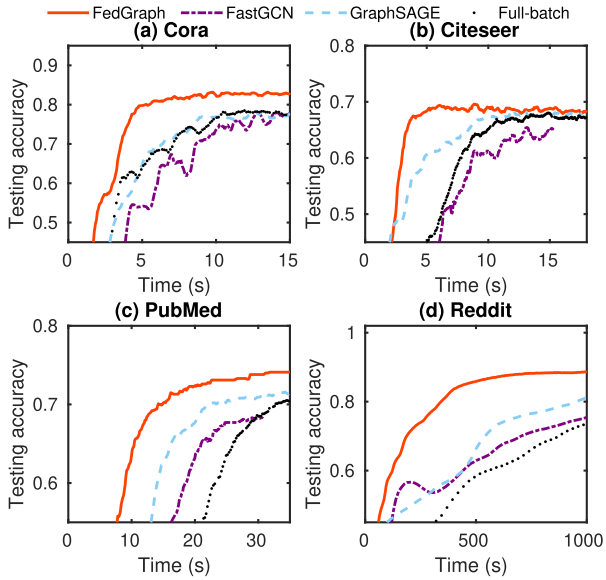

Fig. 7. Accuracy convergence of different sampling schemes with 20 clients. Note that, FastGCN completes the training with less time as it samples fewer nodes for training. However, it has poor performances in all datasets.

the reward function (9), we set the base of exponential function, i.e., Ω, as 128 in our experiments. Since FedGraph relies on the exponential property of reward function, the base has little influence on FedGraph. Moreover, the difference of training accuracy $\lambda[t]$ and target accuracy Λ affects the reward in each round t. For each dataset, we choose the best accuracy reported by existing work. Even we have no knowledge of the best accuracy, we can make an estimation according to experiences. Since FedGraph only relies on the exponential property of reward function, such estimation has little influence to FedGraph. Both constants α and β aim to balance accuracy improvement and time cost. In our experiments, we control the time penalty $\alpha(\delta[t]-\beta)$ close to 1, similar to the settings in [24]. For comparison, we extend the following three graph sampling schemes for federated graph learning.

1) *Full-batch*: We do not conduct graph sampling and use the original graph to construct GCN.
2) *GraphSAGE*: A typical node-wise neighbor-sampling method that iteratively samples a fixed number of neighbors. The neighbor-sampling sizes of two convolutional layers are set as 25 and 10, respectively, which are the same with the settings in [12], [20], [26].
3) *FastGCN*: A typical layer-wise importance-sampling method that independently samples a fixed number of nodes, which is also called layer size, for each layer. The layer size of Cora and Citeseer is set to 256, and that of Reddit and PubMed is 8,192, which are the settings advocated by [21].

In the DRL-based sampling algorithm of FedGraph, both actor-networks and critic-networks have 2 hidden layers of 512 and 256 units. We compress feature weights into 20 dimensions by using the tool `sklearn.decomposition.PCA` [33].

5.2 Experimental Results

Convergence of DRL-Based Sampling. We let FedGraph train 300 episodes and show cumulative returns under four datasets in Fig. 6. We set the target accuracy as 90.16% for Cora, 78.7% for PubMed, 87.9% for Citeseer, and 96.27% for Reddit. We observe that cumulative discounted returns of four datasets can converge to stable values in less than 100 episodes, Especially, the biggest dataset, Reddit, almost converges after 50 episodes, as shown in Fig. 6d. These facts demonstrate good convergence of our proposed DRL-based sampling scheme.

Results of Training Accuracy. The accuracy convergence of different sampling schemes is shown in Fig. 7, where we can see that FedGraph can converge at a faster speed and achieve higher accuracy. For a fair comparison, we use physical time, instead of the number of training rounds, as the metric to evaluate training speeds of different schemes. That is because clients have graphs of different sizes, and they consume different time costs in each training round. Specifically, FedGraph achieves 75% accuracy at about 5 seconds on Cora, but the other three algorithms take more than 10 seconds to achieve similar accuracy. In PubMed, FedGraph takes about 15 seconds to achieve 73% accuracy, but GraphSAGE and full-batch scheme need more than 2 times as long to converge. In the largest datasets Reddit, FedGraph's advantages are more obvious, as shown in Fig. 7d. We summarize the reasons as follows. GraphSAGE has a serious problem of computation redundancy, which consumes more time for training. FastGCN can not get sufficient embedding information from other clients because some sampled nodes have no edge connections. Full-batch scheme needs to calculate the embeddings of all nodes, which incurs high computational cost especially on larger graphs PubMed and Reddit. FedGraph has well addressed the weaknesses of the above methods and thus achieves higher performance. Note that the total number of training rounds is fixed to 300 and FastGCN completes training earlier because it samples fewer nodes for training. Moreover, to evaluate the scalability of FedGraph, we enlarge the experimental scale to 50 clients and show corresponding results in Fig. 8. We can find that FedGraph still outperforms other sampling schemes.

Influence of Graph Heterogeneity. We study the influence of graph heterogeneity by changing the variance of ξ_i. We

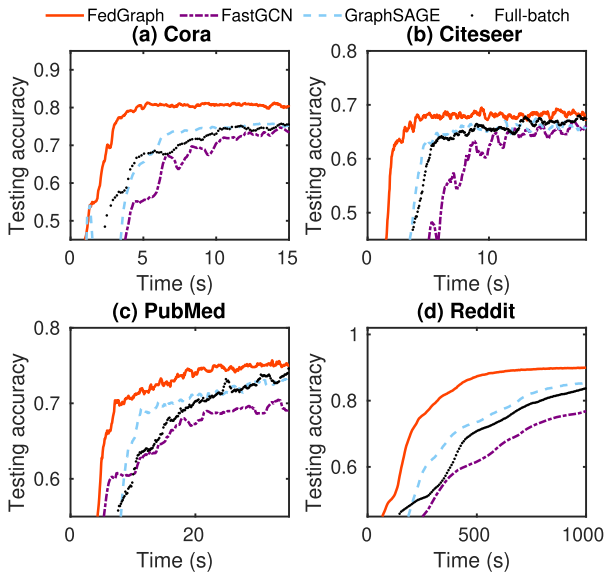

Fig. 8. Accuracy convergence of different sampling schemes with 50 clients.

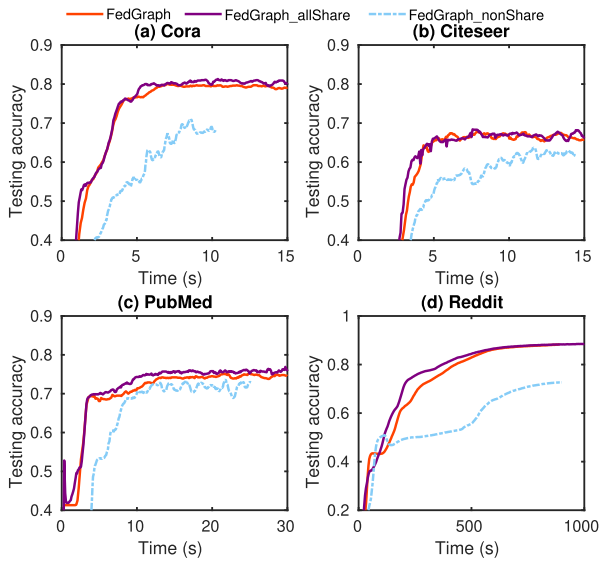

Fig. 10. Convergence of FedGraph and FedGraph_nonShare. FedGraph_nonShare completes the training with less time as it ignores lots of connections in the local training. However, it has a poor convergence.

consider three heterogeneity levels, and the corresponding variances are 0.1 (low), 0.5 (middle) and 1 (high), respectively. For a better understanding, we calculate the ratio between the smallest graph size and the largest size, and the results are about 0.2, 0.4 and 0.6, respectively. We measure the training time to converge to a target accuracy that can be achieved by most of sampling schemes. In PubMed, we set target accuracy to 72%, but FastGCN can converge to 68.6% only. As shown in Fig. 9, the training time of all sampling schemes increases as graphs become more heterogeneous under all datasets. However, FedGraph has better control on the time growth because its DRL-based sampling jointly considers the training speed and accuracy.

Effect of Cross-Client Embedding Sharing. FedGraph uses the cross-client graph convolution operation to enable embedding sharing between clients while hiding local features during local GCN training. For comparison, we consider two alternative methods, one (referred to as FedGraph_allShare) is to share embeddings from the first layer to maximize the information sharing, and the other (referred to as FedGraph_nonShare) is to discard cross-client sharing to simplify the design. We show the accuracy convergence of these three designs in Fig. 10. The total number of training rounds is set to 300. We can find that the curve of FedGraph is close to that of FedGraph_allShare, which demonstrates that FedGraph has little information loss even though it eliminates the embedding sharing in the first layer. It is because that the high-layer embedding contains information about the original features. Hence, FedGraph can efficiently learn from cross-clients embedding sharing without the original feature exchanging. Simultaneously, FedGraph significantly outperforms FedGraph_nonShare under all datasets. In Cora and Citeseer, cross-client convolution operations can increase training accuracy by about 10%. In PubMed, two designs have similar final accuracy, but FedGraph enables quick convergence. Reddit is more sensitive to cross-client embedding sharing than other datasets, and FedGraph_nonShare converges to an accuracy of about 70%, while FedGraph can converge to about 90%. That is because Reddit has rich edge connections as shown in Table 1, and ignoring cross-client edges would seriously break the whole graph structure. Note that FedGraph_nonShare completes 300-round training earlier because it eliminates embedding sharing.

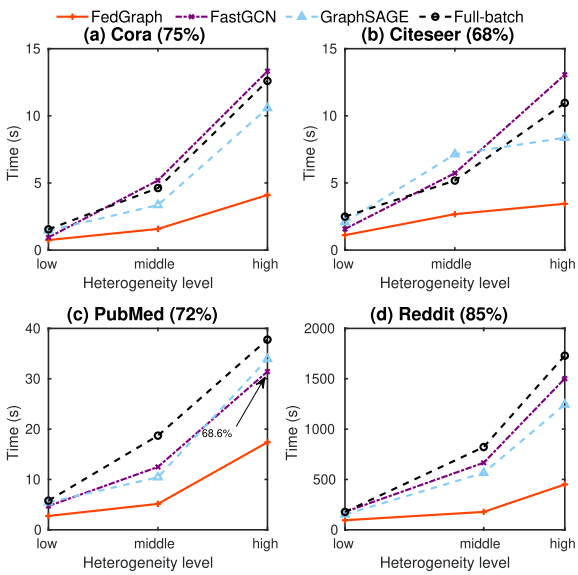

Fig. 9. The convergence time under different levels of graph heterogeneity.

The Impact of GCN Depth. We study the impact of GCN depth by changing the number of graph convolutional layers. The results are shown in Fig. 11. We can see that for all datasets, there is obvious growth of time complexity as we increase the number of layers from 2 to 4. Meanwhile, the accuracy has little changes. In particular, the accuracy of Citeseer decreases as the growth of GCN layers because of the over-smoothing issue [6], [34], [35].

The Impact of Non-IID Data. The effectiveness of FedGraph on handling non-IID data is demonstrated in Fig. 12. We generate the non-iid data distribution by selecting a subset of node types for each local graph. The experimental results show that FedGraph still outperforms other schemes.

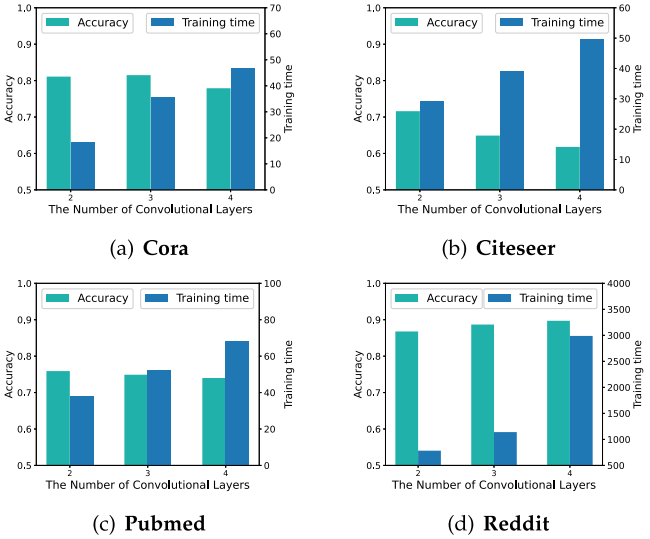

Fig. 11. Training accuracy and time under different GCN depths.

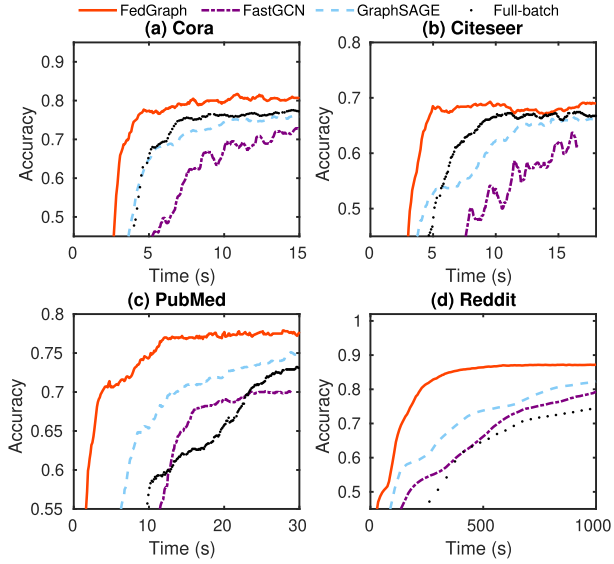

Fig. 12. Accuracy convergence of different sampling schemes on non-iid data.

6 Related Work

6.1 Federated Learning

Federated learning has attracted great research attention due to its great promise in enabling privacy-preserving distributed machine learning [3], [22]. Zhao *et al.* [2] have demonstrated the impact of non-IID data in federated learning with mathematical and proposed an approach that sends a set of uniform distribution data to each client to reduce the effect of non-IID data.

Recently, several works study GNNs under different federated settings from the one in this paper. Suzumura *et al.*, [36] develop a federated learning platform to detect financial crime activities across multiple financial institutions. They extract global graph information to euclidean data by graph analytic methods instead of graph neural networks. Besides, they assume the global graph belongs to all clients. In contrast, we study GCNs on non-euclidean data, and each client owns a local graph.

Jiang *et al.*, [37] propose a novel distributed surveillance system based on GNN and federated learning. There are two critical differences between this work and our paper. First, they consider a cross-device federated setting, involving a large number of cameras with limited computation and communication capability. In contrast, we study a cross-silo federated setting, which typically involves a small number of clients. Second, they aim to protect the trained model. However, we explore inter-client connections and protect node features.

Mei *et al.*, [8] study federated privacy-preserving graph neural networks with a vertical federated setting, i.e., assume that graph structural, features, and labels belong to different sources. However, we consider a horizontal federated setting, i.e., each local client maintains a complete graph dataset with its own graph structure, node features, and labels.

6.2 Graph Convolutional Networks

Due to its excellent performance, GCN has been widely used in many graph learning applications, like node classification [6], [38], link prediction [39], and recommendation systems [40]. Recently, several studies have applied GCN in natural language processing tasks, like machine translation [41] and relation classification [42]. In order to accelerate GCN training, NeuGraph [43] has been proposed as a new framework that supports efficient and scalable parallel neural network computation on graphs. NeuGraph can support not only single GPU training, but also parallel processing on multiple GPUs. Scardapane *et al.* [13] have proposed distributed GCN training based on message passing exchanges. However, this work ignores privacy protection, which is necessary for federated learning scenarios.

Graph sampling can effectively reduce GCN training overhead. Hamilton *et al.* [12] have proposed GraphSAGE that constructs a simplified GCN by sampling a subset of neighboring nodes. However, GraphSAGE incurs redundant computation at some nodes as common neighbors [25]. Although several works has been proposed to alleviate the redundant computation by reducing the size of sampled nodes, like VR-GCN [19] and Cluster-GCN [26], they still can not well address this problem when training a very large and deep GCN. To deal with this problem, layer-wise sampling methods, like FastGCN [20] and LADIES [21], have been proposed to sample the nodes for each layer independently, instead of sampling neighbors for each node. This kind of sampling method can efficiently reduce the computation cost, but some sampled nodes may have no connection due to independent sampling, which would degrade training accuracy. In addition, all above sampling methods depend on hand-crafted parameters that need manual tuning. The weaknesses of existing work motivate the FedGraph design with intelligent sampling in this paper.

7 Conclusion

In this paper, we propose FedGraph as a novel federated graph system to enable privacy-preserving distributed GCN learning. Different from traditional federated learning, FedGraph is more challenging because GCN training process involves embedding sharing among clients. To address this challenge, FedGraph uses a novel cross-client graph

convolution operation to compress the embeddings before sharing, so that private information can be well hidden. In addition, to reduce GCN training overhead, FedGraph adopts a DRL-based sampling scheme that can well balance the training speed and accuracy. Experimental results on a 20-client testbed show that FedGraph significantly outperforms existing schemes.

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. Artif. Intell. Statist. Conf.*, 2017, pp. 1273–1282.

[2] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," 2018, *arXiv:1806.00582*. [Online]. Available: https://arxiv.org/abs/1806.00582

[3] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," in *Proc. NIPS Workshop Private Multi-Party Mach. Learn.*, 2016, pp. 1–10.

[4] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666651021000012

[5] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 4–24, Jan. 2021.

[6] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. Learn. Representations*, 2017, pp. 1–14.

[7] J. Zhou *et al.*, "Privacy-preserving graph neural network for node classification," 2020, *arXiv:2005.11903*. [Online]. Available: https://arxiv.org/abs/2005.11903

[8] G. Mei, Z. Guo, S. Liu, and L. Pan, "SGNN: A graph neural network based federated learning approach by hiding structure," in *Proc. IEEE Int. Conf. Big Data*, 2019, pp. 2560–2568.

[9] C. He *et al.*, "Fedgraphnn: A federated learning system and benchmark for graph neural networks," 2021, *arXiv:2104.07145*. [Online] Available: https://arxiv.org/abs/2104.07145

[10] L. Zheng, J. Zhou, C. Chen, B. Wu, L. Wang, and B. Zhang, "ASFGNN: Automated separated-federated graph neural network.," *Peer-to-Peer Netw. Appl.*, pp. 1–13, 2021.

[11] E. Choi, M. T. Bahadori, L. Song, W. F. Stewart, and J. Sun, "GRAM: graph-based attention model for healthcare representation learning," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2017, pp. 787–795.

[12] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1024–1034.

[13] S. Scardapane, I. Spinelli, and P. Di Lorenzo , "Distributed graph convolutional networks," *IEEE Trans. Signal Inf. Process. over Netw.*, pp. 87–100, 2020.

[14] The Top 20 Valuable Facebook Statistics – Updated August 947, 2020. Accessed: Nov. 20, 2021. [Online]. Available: https://zephoria.com/top-15-valuable-facebook-statistics/

[15] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, "Batchcrypt: Efficient homomorphic encryption for cross-silo federated learning," in *Proc. USENIX Annu. Tech. Conf.*, 2020, pp. 493–506.

[16] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, S. Moriai, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Trans. Inf. Forensics Secur.*, vol. 13, no. 5, pp. 1333–1345, May 2018.

[17] X. Zhang, F. Li, Z. Zhang, Q. Li, C. Wang, and J. Wu, "Enabling execution assurance of federated learning at untrusted particpants," in *Proc. Int. Conf. Comput. Commun.*, 2020, pp. 1877–1886.

[18] T. Lee *et al.*, "Occlumency: Privacy-preserving remote deep-learning inference using SGX," in *Proc. 25th Annu. Int. Conf. Mobile Comput. Netw.*, 2019, pp. 1–17.

[19] J. Chen, J. Zhu, and L. Song, "Stochastic training of graph convolutional networks with variance reduction," in *Proc. Int. Conf. Mach. Learn.*, Stockholm, Sweden, 2018, pp. 942–950.

[20] J. Chen, T. Ma, and C. Xiao, "FastGCN: Fast learning with graph convolutional networks via importance sampling," in *Proc. Int. Conf. Learn. Representations*, 2018, pp. 1–15.

[21] D. Zou, Z. Hu, Y. Wang, S. Jiang, Y. Sun, and Q. Gu, "Layer-dependent importance sampling for training deep and large graph convolutional networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 11249–11259.

[22] J. Konečnỳ, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," 2016, *arXiv:1610.02527*. [Online]. Available: http://arxiv.org/abs/1610.02527

[23] N. H. Tran, W. Bao, A. Zomaya, M. N. H. Nguyen, and C. S. Hong, "Federated learning over wireless networks: Optimization model design and analysis," in *Proc. Int. Conf. Comput. Commun.*, 2019, pp. 1387–1395.

[24] H. Wang, Z. Kaplan, D. Niu, and B. Li, "Optimizing federated learning on non-IID data with reinforcement learning," in *Proc. Int. Conf. Comput. Commun.*, 2020, pp. 1698–1707.

[25] W. Huang, T. Zhang, Y. Rong, and J. Huang, "Adaptive sampling towards fast graph representation learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 4558–4567.

[26] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2019, pp. 257–266.

[27] K. Bonawitz *et al.*"Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 1175–1191.

[28] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 1310–1321.

[29] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," in *Proc. 4th Int. Conf. Learn. Representations*, 2016. [Online]. Available: http://arxiv.org/abs/1509.02971

[30] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics Intell. Lab. Syst.*, vol. 2, no. 1–3, pp. 37–52, 1987.

[31] Deep Graph Library. Accessed: Nov. 20, 2021. [Online]. Available: https://www.dgl.ai/

[32] Z. Cai, X. Yan, Y. Wu, K. Ma, J. Cheng, and F. Yu, "DGCL: An efficient communication library for distributed GNN training," in *Proc. ACM 16th Eur. Conf. Comput. Syst.*, New York, NY, USA, 2021, pp. 130–144.

[33] Fabian Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[34] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," in *Proc. 37th Int. Conf. Mach. Learn.*, 2020, pp. 1725–1735.

[35] Y. Rong, W. Huang, T. Xu, and J. Huang, "Dropedge: Towards deep graph convolutional networks on node classification," in *Proc. 8th Int. Conf. Learn. Representations*, vol. 33, 2020, pp. 1–17. [Online]. Available: https://openreview.net/forum?id=Hkx1qkrKPr

[36] T. Suzumura *et al.*, "Towards federated graph learning for collaborative financial crimes detection," in *Proc. NeurIPS Workshop Robust AI Financial Services: Data, Fairness, Explainability, Trustworthiness, Privacy*, 2019. [Online]. Available: https://arxiv.org/abs/1909.12946

[37] M. Jiang, T. Jung, R. Karl, and T. Zhao, "Federated dynamic gnn with secure aggregation," 2020, *arXiv:2009.07351*. [Online]. Available:https://arxiv.org/abs/2009.0735

[38] X. Liu, Z. Tang, P. Li, S. Guo, X. Fan, and J. Zhang, "A graph learning based approach for identity inference in dapp platform blockchain," *IEEE Trans. Emerging Top. Comput.*, early access, Sep. 29, 2020, doi: 10.1109/TETC.2020.3027309.

[39] T. N. Kipf and M. Welling, "Variational graph auto-encoders," in *Proc. NIPS Workshop Bayesian Deep Learn.*, 2016, pp. 1–3. [Online]. Available:https://arxiv.org/abs/1611.07308

[40] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2018, pp. 974–983.

[41] J. Bastings, I. Titov, W. Aziz, D. Marcheggiani, and K. Sima'an, "Graph convolutional encoders for syntax-aware neural machine translation," in *Proc. Conf. Empirical Methods Nat. Lang. Process.*, 2017, pp. 1957–1967. [Online]. Available: https://www.aclweb.org/anthology/D17-1209

[42] Y. Li, R. Jin, and Y. Luo, "Classifying relations in clinical narratives using segment graph convolutional and recurrent neural networks (SEG-GCRNS)," *J. Amer. Med. Inform. Assoc.*, vol. 26, no. 3, pp. 262–268, 2019.

[43] L. Ma *et al.*, "Neugraph: Parallel deep neural network computation on large graphs," in *Proc. USENIX Annu. Tech. Conf.*, Renton, WA, USA, 2019, pp. 443–458.

Fahao Chen is currently working toward the PhD degree with the Graduate School of Computer Science and Engineering, The University of Aizu, Japan. His research interests include cloud or edge computing and distributed machine learning systems.

Peng Li (Senior Member, IEEE) received the BS degree from the Huazhong University of Science and Technology, China, in 2007, and the MS and PhD degrees from The University of Aizu, Japan, in 2009 and 2012, respectively. He is currently an associate professor with The University of Aizu, Japan. He has authored or coauthored more than 100 technical papers on prestigious journals and conferences. His research interests include cloud or edge computing, Internet-of-Things, machine learning systems, and related wired and wireless networking problems. He is the editor of *IEICE Transactions on Communications* and *IEEE Open Journal of the Computer Society*. He was the recipient of the Young Author Award of IEEE Computer Society Japan Chapter in 2014, the Best Paper Award of IEEE TrustCom 2016, and he supervised students to win the First Prize of IEEE ComSoc Student Competition in 2016.

Toshiaki Miyazaki (Senior Member, IEEE) received the BE and ME degrees in applied electronic engineering from the University of Electro-Communications, Tokyo, Japan, in 1981 and 1983, respectively, and the PhD degree in electronic engineering from the Tokyo Institute of Technology in 1994. He is currently a professor with The University of Aizu, Fukushima, Japan, and the dean of the Undergraduate School of Computer Science and Engineering. He was with NTT for 22 years engaged in research on VLSI CAD systems, telecommunications-oriented FPGAs and their applications, active networks, peer-to-peer communications, and ubiquitous network environments. He was a visiting professor with Graduate School, Niigata University in 2004 and a part-time lecturer with the Tokyo University of Agriculture and Technology from 2003 to 2007. His research interests include reconfigurable hardware systems, adaptive networking technologies, and autonomous systems. He is a member of IEICE and IPSJ.

Celimuge Wu (Senior Member, IEEE) received the ME degree from the Beijing Institute of Technology, China, in 2006 and the PhD degree from The University of Electro-Communications, Japan, in 2010. He is currently an associate professor with the Graduate School of Informatics and Engineering, The University of Electro-Communications. His research interests include vehicular networks, edge computing, IoT, intelligent transport systems, and application of machine learning in wireless networking and computing. He is currently an associate editor for *IEEE Open Journal of the Computer Society*, *IEEE Transactions on Network Science and Engineering*, *IEEE Transactions on Green Communications and Networking*, and *IEEE Access*. He is the chair of IEEE TCGCC Special Interest Group on Green Internet of Vehicles and IEEE TCBD Special Interest Group on Big Data with Computational Intelligence. He was the recipient of the IEEE Computer Society 2019 Best Paper Award Runner-Up.