Subgraph Federated Learning with Missing Neighbor Generation

Ke Zhang [1 2] **Carl Yang** [1 †] **Xiaoxiao Li** [3] **Lichao Sun** [4] **Siu Ming Yiu** [2]

Abstract

Graphs have been widely used in data mining and machine learning due to their unique representation of real-world objects and their interactions. As graphs are getting larger nowadays, it is common to see their subgraphs separately collected and stored in multiple local systems. Therefore, it is natural to consider the *subgraph federated learning* setting, where each local system holding a *small* subgraph that may be *biased* from the distribution of the whole graph. Hence, the subgraph federated learning aims to collaboratively train a powerful and generalizable graph mining model without directly sharing their graph data. In this work, towards the novel yet realistic setting of subgraph federated learning, we propose two major techniques: (1) FedSage, which trains a GraphSage model based on FedAvg to integrate node features, link structures, and task labels on multiple local subgraphs; (2) FedSage+, which trains a missing neighbor generator along FedSage to deal with missing links across local subgraphs. Empirical results on four real-world graph datasets with synthesized subgraph federated learning settings demonstrate the effectiveness and efficiency of our proposed techniques.

1. Introduction

Graph mining leverages links among connected nodes in graphs to conduct inference. Recently, graph neural networks (GNNs) gain applause with impressing performance and generalizability in many graph mining tasks (Wu et al., 2020b; Hamilton et al., 2017; Kipf & Welling, 2017; Luo et al., 2021; Yang et al., 2020a). Similar to machine learning tasks in other domains, attaining a well-performed GNN model requires its training data to not only be sufficient, but also follows the similar distribution as general queries. While in reality, data owners often collect limited and biased graphs and cannot observe the global distribution. Therefore, with heterogeneous subgraphs separately stored in local data owners, accomplishing a globally applicable GNN requires collaboration.

Federated learning (FL) (Li et al., 2020; Yang et al., 2019b), targeting at training machine learning models with data distributed in multiple local systems to resolve the data-silo problem, has shown its advantage in enhancing the performance and generalizability of the collaboratively trained models without the need of sharing any actual data. For example, FL has been devised in computer vision (CV) and natural language processing (NLP) to allow the joint training of powerful and generalizable deep convolutional neural networks and language models on separately stored datasets of images and texts (Liu et al., 2021; Dou et al., 2021; Liang et al., 2019; Zhu et al., 2020b; He et al., 2021b).

Motivating Scenario. Taking the healthcare system as an example, as shown in Fig. 1, residents of a city may go to different hospitals due to various reasons. As a result, their healthcare data, such as demographics and living conditions, as well as patient interactions, such as co-staying in a sickroom and co-diagnosis of a disease, are stored only within the hospitals they visit. When any healthcare problem is to be studied in the whole city, *e.g.*, the prediction of infections when a pandemic occurs, a single powerful graph mining model is needed to conduct effective inference over the whole global patient network, which contains all subgraphs from different hospitals. However, it is difficult to let all hospitals share their patient networks with others to train the graph mining model, due to conflicts of interests.

In such scenarios, it is desirable to train a powerful and generalizable graph mining model over multiple distributed subgraphs without actual data sharing. However, this novel yet realistic setting brings two unique technical challenges, which have never been explored so far.

Challenge 1: How to jointly learn from multiple local subgraphs? In our considered scenario, the global graph is

[1]Department of Computer Science, Emory University, Atlanta, United States [2]Department of Computer Science, The University of Hong Kong, Hong Kong, China [3]Department of Computer Science, Princeton University, Princeton, United States [4]Department of Computer Science, Leigh University, Bethlehem, United States. Correspondence to: Carl Yang <j.carlyang@emory.edu>.

This work was presented at the International Workshop on Federated Learning for User Privacy and Data Confidentiality in Conjunction with ICML 2021 (FL-ICML'21). This workshop does not have official proceedings and this paper is non-archival.

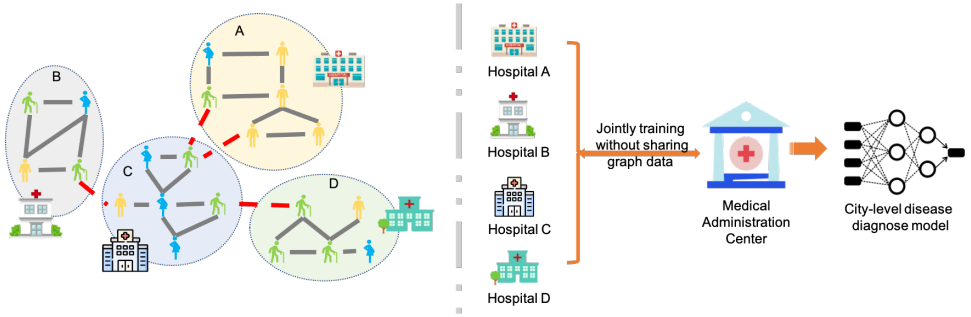


Figure 1. **A toy example of the distributed subgraph storage system:** In this example, there are four hospitals and a medical administration center. The global graph records, for a certain period, the city's patients (nodes), their information (attributes), and interactions (links). Specifically, the left part of the figure shows how the global graph is stored in each hospital, where the grey solid lines are the links explicitly stored in each hospital, and the red dashed lines are the cross-hospital links that may exist but are not stored in any hospital. The right part of the figure indicates our goal that without sharing actual data, the system obtains a globally powerful graph mining model.

distributed into a set of small subgraphs with heterogeneous feature and structure distributions. Training a separate graph mining model on each of them may not capture the global data distribution and is also prone to overfitting. Moreover, it is unclear how multiple graph mining models can be integrated into a universally applicable one that can handle any queries from the underlying global graph.

Solution 1: FedSage: Training GraphSage with FedAvg. To attain a powerful and generalizable graph mining model from small and biased subgraphs distributed in multiple local owners, we develop a framework of subgraph federated learning, specifically, with the vanilla mechanism of FedAvg (McMahan et al., 2017). As for the graph mining model, we resort to GraphSage (Hamilton et al., 2017), due to its advantages of inductiveness and scalability. We term this framework as FedSage.

Challenge 2: How to deal with missing links across local subgraphs? Unlike distributed systems in other domains such CV and NLP, whose data samples of images and texts are isolated and independent, data samples in graphs are connected and correlated. Most importantly, in a subgraph federated learning system, data samples in each subgraph can potentially have connections to those in other subgraphs, which carries important information of node neighborhoods and serves as bridges among the data owners, but they are never directly captured by any data owner.

Solution 2: FedSage+: Generating missing neighbors along FedSage. To deal with cross-subgraph missing links, we propose a novel FedSage+ model on top of FedSage, by adding a missing neighbor generator into the FL framework. Specifically, for each data owner, instead of training the GraphSage model on the entire subgraph, we first impair the subgraph by randomly holding out some nodes and their links; then we jointly train a neighbor generator based on the held-out neighbors to mend the graph and train the GraphSage classifier on the mended graph. This neighbor generator trained on individual local subgraphs can generate potential missing links within the subgraphs, and training it in our subgraph FL setting allows multiple local owners to generate missing neighbors across distributed subgraphs.

We conduct experiments on four real-world datasets with different numbers of data owners to better simulate the application scenarios. Experimental results show both of our models outperform locally trained classifiers in all scenarios. Compared to FedSage, FedSage+ further promotes the outcome classifier. In-depth model analysis shows the convergence and generalization ability of our frameworks.

2. Related works

Graph mining. Graph mining emerges its significance in analyzing the informative graph data, which range from social networks to gene interaction networks (Yang et al., 2021; 2020b; 2019a). One of the most frequently applied tasks on graph data is node classification. Recently, graph neural networks (GNNs), *e.g.*, graph convolutional networks (GCN) (Kipf & Welling, 2017) and GraphSage (Hamilton et al., 2017), improve the state-of-the-art in node classification with their elegant yet powerful designs. However, as GNNs leverage the homophily of nodes in both node features and link structures to conduct the inference, they are vulnerable to the perturbation on graphs (Zügner & Günnemann, 2019). Robust GNNs, aiming at reducing the degeneration in GNNs caused by graph perturbation, are gaining attention these days. Current robust GNNs focus on the sensitivity towards modifications on node features (Chen et al., 2021) or adding/removing edges on the graph (Zhu et al., 2019). However, neither of these two types recapitulates the missing neighbor problem, which affects both the feature distribution and structure distribution.

Moreover, to obtain a node classifier with good generalizability, the development of domain adaptive GNN sheds light on adapting a GNN model trained on the source do-

main to the target domain by leveraging underlying structural consistency (Wu et al., 2020a; Zhu et al., 2020a). In our considered distributed system, however, each data owner has subgraphs with heterogeneous feature and structure distributions, and direct information exchanges, such as message passing among subgraphs, are fully blocked due to the missing cross-subgraph links. The violation to the domain adaptive GNNs' assumptions on alignable nodes and cross-domain structural consistency denies their usage in the distributed subgraph system.

Federated learning. FL is proposed for cross-institutional collaborative learning without sharing raw data (Li et al., 2020; Yang et al., 2019b). FedAvg (McMahan et al., 2017) is an efficient and well-studied FL method. Similar to most FL methods, it is originally proposed for traditional machine learning problems (Yang et al., 2019b) to allow collaborative training on silo data through local updating and global aggregation. Many FL methods are not directly applicable to graph data due to the complex and flexible link structures.

Federated graph learning. Recent researchers have made some progress in federated graph learning. There are existing FL frameworks designed for the graph data learning task (He et al., 2021a; Wu et al., 2021). (He et al., 2021a; Xie et al., 2021) design graph level FL schemes with graph datasets dispersed over multiple data owners, which are inapplicable to our distributed subgraph system construction. (Wu et al., 2021) proposes an FL method for the recommendation problem with each data owner learning on a subgraph of the whole recommendation user-item graph. It considers a different scenario assuming subgraphs have overlapped items (nodes), and the user-item interactions (edges) are distributed but completely stored in the system, which ignores the possible cross-subgraph information lost in real-world scenarios. However, we study a more challenging yet realistic case in the distributed subgraph system, where cross-subgraph edges are totally missing.

In this work, we consider the commonly existing, yet not studied scenario, *i.e.*, distributed subgraph system with missing cross-subgraph edges. Under this scenario, we focus on obtaining a globally applicable node classifier through FL on distributed subgraphs.

3. FedSage

In this section, we first define the distributed subgraph system derived from real-world application scenarios. Based on it, we then formulate our novel subgraph FL framework and a vanilla solution called FedSage.

3.1. Subgraphs Distributed in Local Systems

Notation. We denote a global graph as $G = \{V, E\}$, where V is the node set and E is the edge set. In the FL system, we have a central server S and M data owners with distributed subgraphs. $G_i = \{V_i, E_i\}$ is the subgraph owned by data owner D_i, for $i \in [M]$.

Problem setup. For the whole system, we assume $V = V_1 \cup \cdots \cup V_M$. For simplicity, we also assume no overlapping nodes shared across data owners, namely $V_i \cap V_j = \emptyset$ for $\forall i, j \in [M]$. Note that the central server S only maintains a graph mining model with no actual graph data stored. Any data owner D_i cannot directly retrieve $u \in V_j$ from another data owner D_j. Therefore, for an edge $e_{v,u} \in E$, where $v \in V_i$ and $u \in V_j$, $e_{v,u} \notin E_i \cup E_j$, that is, $e_{v,u}$ might exist in reality but is not stored anywhere in the whole system.

For the global graph $G = \{V, E\}$, every node $v \in V$ has its features $\mathbf{x} \in \mathcal{X}$ and one label $y \in \mathcal{Y}$ for the downstream task, *e.g.*, node classification. For $i \in [M]$, data owner D_i possessing the node set V_i has access to the features and label of each node $v \in V_i$. In a typical GNN, predicting a node's label requires not only the queried node's features, but also its structural information on the graph it belongs to. For a node from graph G with features $\mathcal{X}$, we denote the query made to a node classifier as $q_{G,\mathcal{X}}(\cdot)$ and the query follows the distribution $\mathcal{Q}_{G,\mathcal{X}}$. Thus, a query for node v on G with its ground-truth label y is $q_{G,\mathcal{X}}(v)$ and $(q_{G,\mathcal{X}}(v), y) \sim (\mathcal{Q}_{G,\mathcal{X}}, \mathcal{Y})$.

With subgraphs distributed in the system constructed above, we formulate our goal as follows.

Goal. The system exploits an FL framework $\mathcal{H}_{\mathcal{C}}$ learning on the isolated subgraphs to obtain a global node classifier $\mathcal{C}$, which simulates the distribution of queries drawn from the global graph G without directly seeing G. Formally, we have

$$\begin{aligned}(q_{G,\mathcal{X}}(v), \mathcal{C}\left(q_{G,\mathcal{X}}(v) | \mathcal{H}_{\mathcal{C}}\left(\{G_i, \mathcal{X}_i, \mathcal{Y}_i | i \in [M]\}\right)\right)) \\ \sim \left(\mathcal{Q}_{G,\mathcal{X}}, \mathcal{Y}\right),\end{aligned} \tag{1}$$

where $q_{G,\mathcal{X}}(v)$ is the node classification query generated for node v in G.

3.2. Collaborative Learning on Isolated Subgraphs

To fulfill the system's goal illustrated above, we leverage the simple and efficient FedAvg framework (McMahan et al., 2017) as $\mathcal{H}_{\mathcal{C}}$ and fix the node classifier $\mathcal{C}$ as a GraphSage model, whose inductiveness and scalability concert its training on subgraphs with heterogeneous query distributions and generalization to the global graph. We term this vanilla model as FedSage.

A globally shared K-layer GraphSage classifier $\mathcal{C}$ models the K-hop neighborhood for a queried node $v \in V$ on graph G to conduct prediction with learnable parameters θ_c. Taking G_i as an example, for $v \in V_i$ with features as h_v^0, at each layer $k \in [K]$, $\mathcal{C}$ computes v's representation h_v^k as

$$h_v^k = \sigma\left(\theta^k \cdot \left(h_v^{k-1} || Agg\left(\left\{h_u^{k-1}, \forall u \in \mathcal{N}_{G_i}(v)\right\}\right)\right)\right), \tag{2}$$

where $\mathcal{N}_{G_i}(v)$ is the set of v's neighbors on graph G_i, $||$ is

the concatenation operation, $Agg(\cdot)$ is the aggregator (*e.g.*, mean pooling) and σ is the activation function (*e.g.*, ReLU).

With $\mathcal{C}$ outputting the inference label $\widetilde{y}_v = \text{Softmax}(h_v^K)$ for $v \in V_i$, the supervised loss function $l(\theta|\cdot)$ is defined as

$$\mathcal{L}_c \quad = l(\theta|q_{G_i,\mathcal{X}_i}(v)) = CE(\widetilde{y}_v, y_v) \tag{3}$$

where $CE(\cdot)$ is the cross entropy function, $\theta = \{\theta^k\}_{k=1}^K$ is the set of learnable parameters, $q_{G_i,\mathcal{X}_i(v)}$ contains v's K-hop neighborhood information on G_i, and y_v is the ground-true label of node v.

In FedSage, the distributed subgraph system obtains a shared global node classifier $\mathcal{C}$ parameterized by θ_c through e_c epochs of training. During each epoch t, every D_i first locally computes $\theta_c^{(i)} \leftarrow \theta_c - \eta\nabla\ell(\theta_c|\{(q_{G_i,\mathcal{X}_i}(v), y_v)|v \in V_i^t\})$, where $V_i^t \subseteq V_i$, y_v is the true label of v, and η is the learning rate; then the central server S collects the latest $\{\theta_c^{(i)}|i \in [M]\}$; next, through averaging over $\{\theta_c^{(i)}|i \in [M]\}$, S sets θ_c as the averaged value; finally, S broadcasts θ_c to data owners and finishes one round of training $\mathcal{C}$. After e_c epochs, the distributed subgraph system retrieves $\mathcal{C}$ as the globally useful classifier, which is not limited to or biased towards the queries in any specific data owner.

Unlike FL on Euclidean data, nodes in subgraphs distributed in multiple data owners can potentially interact with each other in reality, but the cross-subgraph links cannot be captured by any data owner. Incomplete neighborhoods in subgraphs prevent the global classifier $\mathcal{C}$ from capturing the true global query distribution.

4. FedSage+

In this section, we propose a novel framework of FedSage+, *i.e.*, subgraph FL with missing neighbor generation. We first design a missing neighbor generator (NeighGen) and its training schema via graph mending. Then, we describe the joint training of NeighGen and GraphSage to better achieve the goal in Eq. (1). WLOG, we demonstrate NeighGen$_i$, *i.e.*, the missing neighbor generator of D_i, as an example, where $i \in [M]$.

4.1. Missing Neighbor Generator (NeighGen)

Graph mending simulation. For our system, we assume that each data owner has missing links only to a particular set of nodes that belong to other data owners. The assumption is realistic yet non-trivial for it both seizing the quiddity of the distributed subgraph system, and allowing us to locally simulate the missing neighbor situation through a graph impairing and mending process. Specifically, in each local graph G_i, we randomly hold out $h\%$ of its nodes $V_i^h \subset V_i$ and all links involving them E_i^h , to form a subgraph, denoted as $\bar{G}_i$, with the impaired set of nodes $\bar{V}_i = V_i \setminus V_i^h$, and edges $\bar{E}_i = E_i \setminus E_i^h$. Then we simulate a graph mending process to train a missing neighbor generator (NeighGen) on the impaired graph $\bar{G}_i = \{\bar{V}_i, \bar{E}_i\}$.

Neural architecture of NeighGen. As shown in Fig. 2, NeighGen consists of two modules, *i.e.*, an encoder $\mathcal{E}$ and a generator $\mathcal{G}$. We describe their designs in details below.

$\mathcal{E}$: A GNN model, *i.e.*, a K-layer GraphSage encoder, with parameters θ_e. For node $v \in \bar{V}_i$ on the input impaired graph $\bar{G}_i$, $\mathcal{E}$ computes node embeddings $z = h^K$ according to Eq. (2) by substituting θ, G with θ_e and $\bar{G}_i$.

$\mathcal{G}$: A generative model recovering missing neighbors for the input graph based on the node embedding z. $\mathcal{G}$ contains dGen and fGen, where dGen is a linear regression model parameterized by θ_d that predicts the number of missing neighbors $\widetilde{n}_i$, and fGen is a feature generator parameterized by θ_f that generates a set of $\widetilde{n}_i$ feature vectors $\widetilde{X}_i$. Both dGen and fGen are constructed as fully connected neural networks (FNNs), while fGen is further equipped with a Gaussian noise generator $N(0,1)$ and a random sampler $\mathcal{R}$ that make it variational and able to generate a set of diverse neighbor features from a single node. Thus, we have

$$\widetilde{n}_i = \sigma(\theta_d \cdot z_i), \text{ and } \widetilde{X}_i = \sigma\left(\theta_f \cdot \mathcal{R}(z_i + N(0,1), \widetilde{n}^i)\right). \tag{4}$$

Accordingly, the training of NeighGen boils down to jointly training dGen and fGen with

$$\begin{aligned} \mathcal{L}_n = \quad & \lambda_d\mathcal{L}_d + \lambda_f\mathcal{L}_f \\ = \quad & \lambda_d \tfrac{1}{|\bar{V}|}\textstyle\sum_{v\in\bar{V}} L_1^S(\widetilde{n}_v - n_v) \\ & +\lambda_f \tfrac{1}{|\bar{V}|}\textstyle\sum_{v\in\bar{V}}\sum_{p\in[\widetilde{n}_v]} \min_{q\in[n_v]}(||\widetilde{x}_v^p - x_v^q||_2^2), \end{aligned} \tag{5}$$

where L_1^S is the smooth L1 distance (Girshick, 2015), n_v and X_v are retrieved based on the hidden nodes V^h.

Obtaining a mended graph G_i' from G_i requires two steps, which are also shown in Fig. 2: 1) Training NeighGen on the impaired graph $\bar{G}_i$ w.r.t. the ground-truth hidden neighbors V_i^h, 2) Referring to the relation between $\bar{G}_i$ and G_i, further mending the original graph G_i into G_i' by running the learned NeighGen on G_i. On the local graph G_i alone, this process can be understood as a data augmentation that further generates potential missing neighbors within G_i. However, the actual goal is to allow NeighGen to generate the cross-subgraph missing neighbors, which can be achieved via training NeighGen with FL and will be discussed in Section 4.3.

4.2. Local Joint Training of GraphSage and NeighGen

While NeighGen is designed to recover missing neighbors, the final goal of our system is to train a node classifier. Therefore, we design the joint training of GraphSage and NeighGen, which leverages neighbors generated by NeighGen to assist the node classification by GraphSage. We term

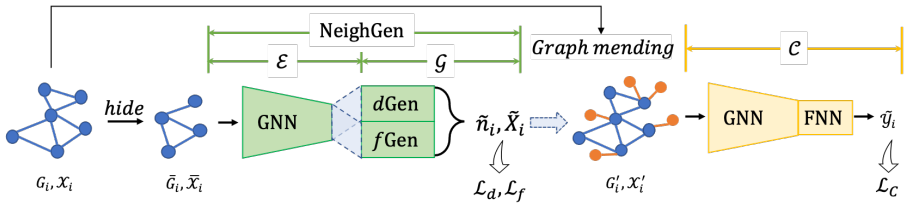


Figure 2. Joint training of missing neighbor generation and node classification.

the integration of GraphSage and NeighGen on the local graphs as LocSage+.

After NeighGen mends the graph G_i into G_i', the GraphSage classifier $\mathcal{C}$ is applied on G_i', accroding to Eq. (2) (with G_i replaced by G_i'). The joint training of NeighGen and GraphSage is done through optimizing the loss function as

$$\mathcal{L} = \mathcal{L}_n + \lambda_c \mathcal{L}_c = \lambda_d \mathcal{L}_d + \lambda_f \mathcal{L}_f + \lambda_c \mathcal{L}_c, \tag{6}$$

where $\mathcal{L}_d$ and $\mathcal{L}_f$ are defined in Eq. (5) and $\mathcal{L}_c$ is defined in Eq. (3) (with G_i substituted by G_i').

However, like GraphSage, the information encoded in the local NeighGen is limited to and biased towards the local graph. Thus, it is natural to also train NeighGen with FL.

4.3. Federated Learning of GraphSage and NeighGen

We observe that cooperation through directly averaging weights of NeighGen across the system can negatively effect its performance. Therefore, instead of training a single centralized NeighGen in a FedAvg manner, we train a local NeighGen$_i$ for each data owner D_i. To allow each NeighGen$_i$ to generate diverse neighbors similar to those missed into other subgraphs $G_j, j \in [M] \setminus \{i\}$, we add a cross-subgraph feature reconstruction loss into fGen$_i$ as:

$$\begin{aligned}\mathcal{L}_{f,i} =& \frac{1}{|\bar{V}_i|} \sum_{v \in \bar{V}_i} \sum_{p \in [\tilde{n}_v]} \min_{q \in [n_v]} (||\tilde{x}_v^p - x_v^q||_2^2) \\ &+ \frac{\alpha}{|\bar{V}_i|} \sum_{v \in \bar{V}_i} \sum_{p \in [\tilde{n}_v]} \sum_{j \in [M]/i} \min_{v_q \in V_j^h} (||\tilde{x}_v^p - x^q||_2^2),\end{aligned} \tag{7}$$

where $v_q \in V_j^h$, $\forall j \in [M] \setminus \{i\}$ is picked as the closest node from the set of hidden nodes in each subgraph G_j to simulate the neighbor of $v \in V_i$ missed into G_j.

Through Eq. (7), NeighGen$_i$ is expected to perceive diverse neighborhood information from all data owners, so as to generate more realistic cross-subgraph missing neighbors.

5. Experiments

We conduct experiments on four datasets to verify the effectiveness of FedSage and FedSage+, under different testing scenarios. We further provide case studies visualizing how they assist the learning process.

5.1. Datasets and experimental settings

We synthesize the distributed subgraph system with four widely used real-world graph datasets, *i.e.*, Cora, Citeseer (Sen et al., 2008), PubMed (Namata et al., 2012), and MSAcademic (Shchur et al., 2018). To synthesize the distributed subgraph system, we find hierarchical graph clusters on each dataset with the Louvain algorithm (Blondel et al., 2008) and use the clustering results with 3, 5, and 10 clusters of similar sizes to obtain subgraphs for data owners.

We implement GraphSage with two layers and mean aggregator. We set the batch size as 64, and the number of nodes sampled in each layer as 5. The training-validation-testing ratio is 60%-20%-20%. The graph impairing ratio varies for different scenarios ($h\% \in [5\%, 15\%]$). All λs are simply set to 1. Optimization is done with Adam with learning rate 0.001. Local training epoch is 1 for each communication round of both FedSage and FedSage+. We implement FedSage and FedSage+ in Python, and execute all experiments on a serve with 8 NVIDIA GeForce GTX 1080 Ti GPUs.

Since we are the first to study the novel yet important setting of subgraph FL, there are no existing baselines. We conduct comprehensive ablation evaluation by comparing FedSage and FedSage+ with three models, *i.e.*, 1) GlobSage: the GraphSage model trained on the original global graph without missing links (as an upper bound), 2) LocSage: one GraphSage model trained solely on each subgraph, 3) LocSage+: the GraphSage plus NeighGen model jointly trained solely on each subgraph.

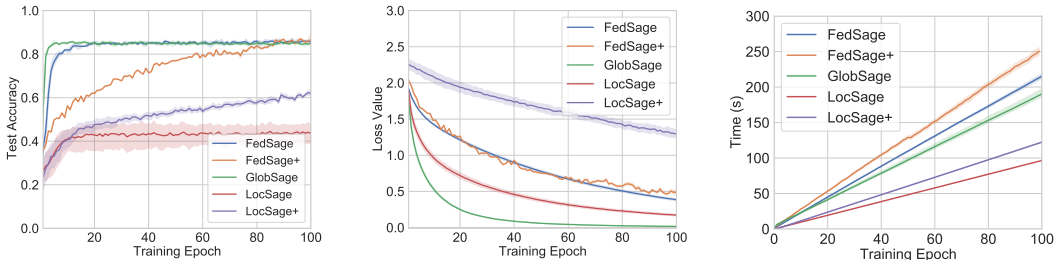
The metric used in our experiments is the node classification accuracy on the queries sampled from the testing nodes on the global graph. For globally shared models of GlobSage, FedSage, and FedSage+, we report the average accuracy over five random repetitions, while for locally possessed models of LocSage and LocSage+, the scores are further averaged across local models.

5.2. Experimental results

Overall performance. We conduct comprehensive ablation experiments to verify the significant promotion brought by FedSage and FedSage+ for local owners in global node classification, and the results are listed in Table 1. The most striking observation emerging from the results is that FedSage+ remarkably outperforms LocSage by at most 46.68%

Table 1. Node classification results on four datasets with M = 3, 5, and 10. Corresponding std values are provided.

	Cora			Citesser			PubMed			MSAcademic		
Model	M=3	M=5	M=10	M=3	M=5	M=10	M=3	M=5	M=10	M=3	M=5	M=10
LocSage	0.5762 ±0.0302	0.4431 ±0.0847	0.2798 ±0.0080	0.6789 ±0.054	0.5612 ±0.086	0.4240 ±0.0859	0.8447 ±0.0047	0.8039 ±0.0337	0.7148 ±0.0951	0.8188 ±0.0331	0.7426 ±0.0790	0.5918 ±0.1005
LocSage+	0.5644 ±0.0219	0.4533 ±0.047	0.2851 ±0.0080	0.6848 ±0.0517	0.5676 ±0.0714	0.4323 ±0.0715	0.8481 ±0.0041	0.8046 ±0.0318	0.7039 ±0.0925	0.8393 ±0.0330	0.7480 ±0.0810	0.5927 ±0.1094
FedSage	0.9071 ±0.0025	0.8968 ±0.0012	0.4917 ±0.0080	0.8499 ±0.0013	0.8192 ±0.0012	0.8192 ±0.0018	0.8238 ±0.0004	0.8046 ±0.0003	0.7742 ±0.0006	0.9327 ±0.0005	0.9391 ±0.0007	0.9262 ±0.0009
FedSage+	**0.9269** ±0.0011	**0.9099** ±0.0013	**0.5505** ±0.0080	**0.8526** ±0.0007	**0.8272** ±0.0010	**0.8269** ±0.0013	**0.8716** ±0.0004	**0.8279** ±0.0004	**0.8285** ±0.0010	**0.9359** ±0.0005	**0.9414** ±0.0006	**0.9314** ±0.0009
GlobSage	0.9213±0.0010			0.8554±0.0010			0.9342±0.0009			0.9681±0.0006		



(a) Accuracy curves (b) Loss curves (c) Training time

Figure 3. Training curves of different frameworks (GlobSage provides an upper bound).

and the vanilla FedSage by at most 5.88% (absolute accuracy gain). Notably, for the Cora dataset, when M is 3, FedSage+ even exceeds GlobSage by 0.56% .

The large gaps between locally obtained classifier, *i.e.*, through LocSage or LocSage+, and the federated trained classifier, *i.e.*, with FedSage or FedSage+, assay the benefits brought by the collaboration across data owners in our distributed subgraph system. Compared to FedSage, the further elevation brought by FedSage+ corroborates the assumed degeneration brought by missing cross-subgraph links and the effectiveness of our innovatively designed NeighGen module. Note that, the gaps between LocSage and LocSage+ are comparatively smaller, indicating that our NeighGen serves more than a robust GNN trainer, but is rather uniquely crucial in the subgraph FL setting.

Case studies. To further understand how FedSage improves the global classifier over LocSage, we provide case study results on Cora with five data owners in Fig. 3. With the assistance of NeighGen, FedSage+ reaches a even higher testing accuracy compared to GlobSage. While FedSage can consistently achieve convergence with rapidly improved testing accuracy similar to GlobSage. Regarding runtime, even though the classifier from FedSage+ learns from distributed mended subgraphs, FedSage+ does not consume observable more training time compared to FedSage. Due to the additional communications and computations in subgraph FL, both FedSage and FedSage+ consume slightly more training time compared to GlobSage.

6. Conclusion

This work aims at obtaining a generalized node classification model in a distributed subgraph system without data sharing. To resolve the limitation in data accessibility, we interweave GraphSage and FedAvg and propose a federated graph learning method, FedSage. To tackle the realistic yet unexplored issue of missing cross-subgraph links, we design a novel missing neighbor generator NeighGen with the corresponding local and federated training processes. Combining NeighGen with FedSage, we present FedSage+. Experimental results evidence the distinguished elevation brought by FedSage and FedSage+ by allowing local data owners to collaboratively learn a global node classifier. Notably, FedSage+ exceeds all compared methodologies in all testing scenarios, which indicates it a practical and universal solution in real-world applications.

Though FedSage+ is manifested with advantageous performance, similar to existing FL methods, it confronts the potential adversarial analysis during interactions. As communications are vital for FL, leveraging cryptologic techniques to minimize the privacy leakage risk in the distributed subgraph system can be a promising future direction.

References

Blondel, V. D., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E. Fast unfolding of communities in large networks. *JSTAT*, 2008(10):P10008, 2008.

Chen, L., Li, J., Peng, Q., Liu, Y., Zheng, Z., and Yang, C. Understanding structural vulnerability in graph convolutional networks. In *IJCAI*, 2021.

Dou, Q., So, T. Y., Jiang, M., Liu, Q., Vardhanabhuti, V., Kaissis, G., Li, Z., Si, W., Lee, H. H., Yu, K., et al. Federated deep learning for detecting covid-19 lung abnormalities in ct: a privacy-preserving multinational validation study. *NPJ digital medicine*, 4:1–11, 2021.

Girshick, R. Fast r-cnn. In *ICCV*, 2015.

Hamilton, W. L., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *NeurIPS*, 2017.

He, C., Balasubramanian, K., Ceyani, E., Rong, Y., Zhao, P., Huang, J., Annavaram, M., and Avestimehr, S. Fedgraphnn: A federated learning system and benchmark for graph neural networks. *arXiv preprint arXiv:2104.07145*, 2021a.

He, C., Li, S., Soltanolkotabi, M., and Avestimehr, S. Pipetransformer: Automated elastic pipelining for distributed training of transformers. *arXiv preprint arXiv:2102.03161*, 2021b.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Li, T., Sahu, A. K., Talwalkar, A., and Smith, V. Federated learning: Challenges, methods, and future directions. *IEEE SPM*, 37:50–60, 2020.

Liang, X., Liu, Y., Chen, T., Liu, M., and Yang, Q. Federated transfer reinforcement learning for autonomous driving. *arXiv preprint arXiv:1910.06001*, 2019.

Liu, Q., Chen, C., Qin, J., Dou, Q., and Heng, P.-A. Feddg: Federated domain generalization on medical image segmentation via episodic learning in continuous frequency space. *arXiv preprint arXiv:2103.06030*, 2021.

Luo, G., Li, J., Peng, H., Yang, C., Sun, L., Yu, P., and He, L. Graph entropy guided node embedding dimension selection for graph neural networks. In *IJCAI*, 2021.

McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, 2017.

Namata, G., London, B., Getoor, L., and Huang, B. Query-driven active surveying for collective classification. In *MLG workshop*, 2012.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Shchur, O., Mumme, M., Bojchevski, A., and Günnemann, S. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

Wu, C., Wu, F., Cao, Y., Huang, Y., and Xie, X. Fedgnn: Federated graph neural network for privacy-preserving recommendation. *arXiv preprint arXiv:2102.04925*, 2021.

Wu, M., Pan, S., Zhou, C., Chang, X., and Zhu, X. Unsupervised domain adaptive graph convolutional networks. In *WWW*, 2020a.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. A comprehensive survey on graph neural networks. *TNNLS*, 2020b.

Xie, H., Ma, J., Xiong, L., and Yang, C. Federated graph classification over non-iid graphs. In *ICML-FL*, 2021.

Yang, C., Zhuang, P., Shi, W., Luu, A., and Li, P. Conditional structure generation through graph variational generative adversarial nets. In *NIPS*, 2019a.

Yang, C., Xiao, Y., Zhang, Y., Sun, Y., and Han, J. Heterogeneous network representation learning: A unified framework with survey and benchmark. In *TKDE*, 2020a.

Yang, C., Zhang, J., and Han, J. Co-embedding network nodes and hierarchical labels with taxonomy based generative adversarial nets. In *ICDM*, 2020b.

Yang, C., Wang, H., Zhang, K., Chen, L., and Sun, L. Secure deep graph generation with link differential privacy. In *IJCAI*, 2021.

Yang, Q., Liu, Y., Chen, T., and Tong, Y. Federated machine learning: Concept and applications. *TIST*, 10(2):1–19, 2019b.

Zhu, D., Zhang, Z., Cui, P., and Zhu, W. Robust graph convolutional networks against adversarial attacks. In *SIGKDD*, 2019.

Zhu, Q., Xu, Y., Wang, H., Zhang, C., Han, J., and Yang, C. Transfer learning of graph neural networks with ego-graph information maximization. *arXiv preprint arXiv:2009.05204*, 2020a.

Zhu, X., Wang, J., Hong, Z., and Xiao, J. Empirical studies of institutional federated learning for natural language processing. In *EMNLP*, pp. 625–634, 2020b.

Zügner, D. and Günnemann, S. Adversarial attacks on graph neural networks via meta learning. In *ICLR*, 2019.