Subgraph Federated Learning with Missing Neighbor Generation

Anonymous Author(s)
Affiliation
Address
email

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 bigger and bigger 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 those local systems, each holding a *small* subgraph that may be *biased* from the distribution of the whole graph, aim 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. At the same time, consistent theoretical implications are made towards their generalization ability on the global graphs.

1 Introduction

Graph mining leverages links among connected nodes in graphs to conduct inference [1]. Recently, graph neural networks (GNNs) gain applause with impressing performance and generalizability in many graph mining tasks [2, 3, 4]. 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 follow 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) [5, 6], targeting at training machine learning models with data distributed in multiple local systems to resolve the information-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 [7, 8, 9, 10, 11].

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 underlying global patient network, which contains all subgraphs from different hospitals.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

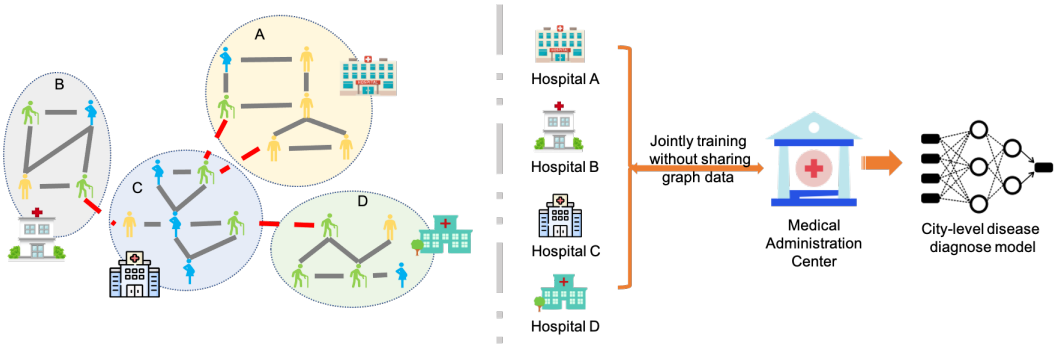


Figure 1: **A toy example of the distributed subgraph storage system:** In this example, there are four hospitals and a medical administration center. The global graph records, for a certain period, the city's patients (nodes), their information (attributes), and interactions (links). Specifically, the 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.

However, it is rather difficult to let all hospitals share their patient networks with each other to train the graph mining model, due to data privacy and conflicts of interests.

There are two unique challenges in training a powerful and generalizable graph mining model over multiple distributed subgraphs without actual data sharing, 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 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 [12]. As for the graph mining model, we resort to GraphSage [3], 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 a local subgraph thus can generate potential missing links in the testing phase, and training it in our subgraph FL setting allows local owner to generate missing neighbors across subgraphs.

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

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 [1]. One of the most frequently applied tasks on graph data is node classification. Recently, graph neural networks (GNNs), *e.g.*, GraphSage [3] and graph convolutional networks (GCN) [4] , improve the state-of-the-art in node classification with their sophisticated designs. However, as GNNs leverage the homophily of nodes in both feature and structure to conduct the inference, GNNs are vulnerable to the perturbation on graphs [13, 14, 15]. 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 [16, 17] or adding/removing edges on the graph [18]. However, neither of these two types recapitulates the neighbors missing problem, which affects both the features distribution and the graph structures.

Moreover, to obtain a node classifier with generalizability, the development of domain adaptive GNN sheds light on adapting a GNN model trained on the source domain to the target domain by leveraging underlying structural consistency [19, 20]. In our considered distributed system, however, each data owner has heterogeneous subgraphs due to unpredictable missing cross-subgraph links, which bring diverse local structures. The violation for the domain adaptive GNN assumptions on cross-domain structural consistency denies its usage in the distributed subgraph system.

Federated learning. FL is proposed for cross-institutional collaborative learning without sharing raw data [5, 6, 12]. FedAvg [12] is an efficient and well-studied FL method, similar to most FL methods, is originally proposed for traditional machine learning problems [6] to allow collaborative training on silo data through local updating and global aggregation. In the distributed subgraph system, to obtain a globally applicable model without sharing local graph data, we borrow the idea of FL to jointly train a GNN.

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 [21, 22]. [21] designs a graph level FL scheme with graph datasets dispersed over multiple data owners, which is inapplicable to our distributed subgraph system construction. While [22] proposes an FL method for the recommendation problem with each data owner learning on a subgraph of the whole recommendation user-item graph. However, [22] considers a different distributed scenario with assumeing subgraphs have overlapped items (nodes), and the user-item interactions (edges) are distributed but completely stored in the system, while our distributed subgraph system fails in recording the cross-subgraph edges.

In this work, we consider the commonly existing, yet not been 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 subgraphs.

3 FedSage

In this section, we first illustrate the construction of the distributed subgraph system derived from real-world application scenarios. Based on this system, 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 denote the central server S, and M data owners in this distributed subgraph system. $G_i = \{V_i, E_i\}$ is the subgraph owned by client 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 as 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 a same 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 spatial information on the graph it belongs to. For a node from graph g with features x, we denote the query made to a node classifier as $q_{g,x}(\cdot)$ and the query follows the distribution $\mathcal{Q}_{g,x}$. 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 a FL framework $\mathcal{H}_{\mathcal{C}}$ to collaboratively learn on the isolated subgraphs in all data owners to obtain a global node classifier $\mathcal{C}$ optimized for queries following the distribution of queries on the global graph G as

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

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 [12] 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 inner parameters θ_c. Taking $G = G_i$ 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 follows

$$\mathcal{L}_c = l(\theta | q_{G_i,\mathcal{X}_i}(v)) = CE(\widetilde{y}_v, y_v) = -\left[y_v \log \widetilde{y}_v + (1 - y_v) \log\left(1 - \widetilde{y}_v\right)\right], \tag{3}$$

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. Still, ignorance of such missing links makes the neighborhoods of nodes in each subgraph incomplete, which prevents 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 jointly training of NeighGen and GraphSage to better achieve the goal in Eq. (1). Without loss of generality, in the following demonstration, we take NeighGen$_i$, *i.e.*, the missing neighbor generator of D_i, as an example, where $i \in [M]$.

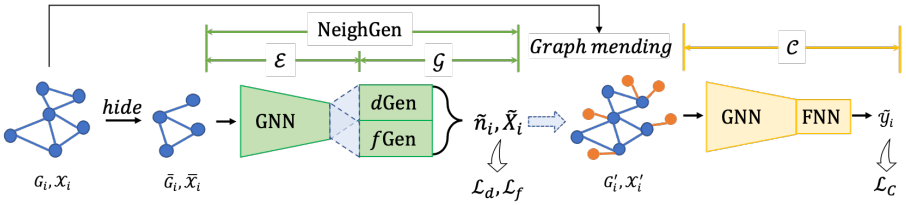


Figure 2: Joint training of missing neighbor generation and node classification.

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 $E_i^h = \{e_{uv} | u \in V_i^h \text{ or } v \in V_i^h\} \subset E_i$, 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 $\bar{G}_i = \{\bar{V}_i, \bar{E}_i\}$ based on the ground-truth missing nodes V_i^h and links E_i^h.

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 in the following.

$\mathcal{E}$: 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

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

where L_1^S is the smooth L1 distance [23], n_v and X_v are retrieved based on the hidden nodes V^h.

To obtain a mended graph G_i' from G_i, a data owner D_i performs 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. However, the actual goal is to train NeighGen through federated learning and allows it to generate the cross-subgraph missing neighbors (links), which will become clear later.

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 the integration of GraphSage and NeighGen on the local graphs 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'). Thus, the joint training of NeighGen and GraphSage is done

through optimizing the following loss function

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

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').

The local joint training of GraphSage and NeighGen allows NeighGen to generate missing neighbors in the local graph that are helpful for the classifications made by GraphSage. However, like GraphSage, the information encoded in the local NeighGen is limited to and biased towards the local graph, which does not enable it to really generate neighbors belonging to other data owners connected by the missing cross-subgraph links. To this end, it is natural to also train NeighGen with federated learning.

4.3 Federated Learning of GraphSage and NeighGen

Similarly to GraphSage alone as described in Section 3.2, we can apply FedAvg to the joint training of GraphSage and NeighGen, by setting the loss function to $\mathcal{L}$ and learnable parameters to $\{\theta_e; \theta_d; \theta_f; \theta_c\}$. However, we observe that cooperation through directly averaging weights of NeighGen across the system can negatively effect its performance, *i.e.*, averaging the weights of a single NeighGen model does not really allow it to generate diverse neighbors from different subgraphs. Recalling our goal of constructing NeighGen, which is to facilitate the training of a centralized GraphSage classifier by generating diverse missing neighbors in each subgraph, we do not necessarily need a centralized NeighGen. Therefore, instead of training a single centralized NeighGen, we train a local NeighGen$_i$ for each data owner D_i. In order 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 follows:

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

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 other than G_i 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. The expectedly diverse and unbiased neighbors further assist the FedSage in training a globally useful classifier that satisfies our goal in Eq. (1). Note that, the additional communications and computation time incurred by Eq. (7) are acceptable since the size of hidden nodes V^h in each subgraph is often pretty small (*e.g.*, 5-15% in our experiments). We list the full pseudo code of FedSage+ with detailed server and client procedures in Alg. 1 in the Appendix.

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 local data owners in accommodating queries from the global distribution.

5.1 Datasets and experimental settings

We synthesize the distributed subgraph system with four widely used real-world graph datasets, *i.e.*, Cora [24], Citeseer [24], PubMed [25], and MSAcademic [26]. To synthesize the distributed subgraph system, we find hierarchical graph clusters on each dataset with the Louvain algorithm [27] and use the clustering results with 3, 5, and 10 clusters of similar sizes to obtain subgraphs for data owners. The statistics of these datasets are presented in Table 1.

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. All λs are simply set to 1. Adam is selected as the optimization algorithm, and its learning rate is 0.001. We implement FedSage and FedSage+ in Python, and execute all experiments on NVIDIA GeForce GTX 1080 Ti GPU.

Since we are the first to study the novel yet important setting of subgraph federated learning, there are no existing baselines. We conduct comprehensive ablation evaluation by comparing FedSage and

Table 1: Statistics of the datasets and the synthesized distributed subgraph systems with M = 3, 5, and 10. #C row shows the number of classes, $|V_i|$ and $|E_i|$ rows show the averaged numbers of nodes and links in all subgraphs, and ΔE shows the total number of missing cross-subgraph links.

Data	Cora			Citeseer			PubMed			MSAcademic		
#C	7			6			3			15		
$\|V\|$	2708			3312			19717			18333		
$\|E\|$	5429			4715			44338			81894		
M	3	5	10	3	5	10	3	5	10	3	5	10
$\|V_i\|$	903	542	271	1104	662	331	6572	3943	1972	6111	3667	1833
$\|E_i\|$	1675	968	450	1518	902	442	12932	7630	3789	23584	13949	5915
ΔE	403	589	929	161	206	300	5543	6189	6445	11141	12151	22743

Table 2: Node classification results on four datasets with M = 3, 5, and 10. Besides averaged accuracy, we also provide the corresponding std. Unspecified std values are all under 10^{-4}.

	Cora			Citesser		
Model	M=3	M=5	M=10	M=3	M=5	M=10
LocSage	0.5762 (±0.0014)	0.4431 (±0.0072)	0.2798 (±0.0080)	0.6789 (±0.0029)	0.5612 (±0.0074)	0.4240 (±0.0074)
LocSage+	0.5644 (±0.0007)	0.4533 (±0.0022)	0.2851 (±0.0080)	0.6848 (±0.0027)	0.5676 (±0.0051)	0.4323 (±0.0051)
FedSage	0.9071	0.8968	0.4917	0.8499	0.8192	0.8192
FedSage+	**0.9269**	**0.9099**	**0.5505**	**0.8526**	**0.8272**	**0.8269**
GlobSage	0.9213			0.8554		
	PubMed			MSAcademic		
Model	M=3	M=5	M=10	M=3	M=5	M=10
LocSage	0.8447 (±0.0000)	0.8039 (±0.0011)	0.7148 (±0.0090)	0.8188 (±0.0016)	0.7426 (±0.0062)	0.5918 (±0.0101)
LocSage+	0.8481 (±0.0000)	0.8046 (±0.0010)	0.7039 (±0.0085)	0.8393 (±0.0016)	0.7480 (±0.0066)	0.5927 (±0.0120)
FedSage	0.8238	0.8046	0.7742	0.9327	0.9373	0.9262
FedSage+	**0.8716**	**0.8279**	**0.8285**	**0.9359**	**0.9422**	**0.9314**
GlobSage	0.9342			0.9681		

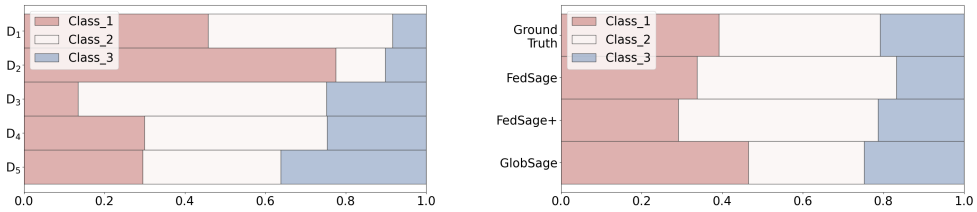
FedSage+ with three models, *i.e.*, 1) GlobSage: the GraphSage model trained on the original global graph without missing links (as an upper bound for FL framework with GraphSage model alone), 2) LocSage: one GraphSage model trained solely on each subgraph, 3) LocSage+: the GraphSage plus NeighGen model 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 scores with variance 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

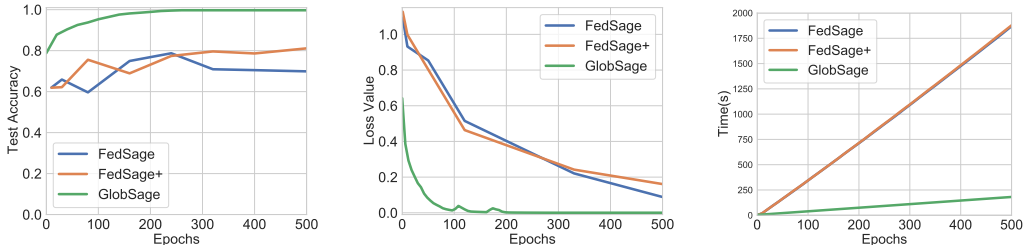
Overall performance. We conduct comprehensive ablation experiments to verify the significant promotion brought by FedSage and FedSage+ for local owners in global node classification, and the results are listed in Table 2. The most striking observation emerging from the results is that FedSage+ remarkably outperforms LocSage by at most 46.68% 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,



(a) Ground-truth on different local data owners (b) Global ground-truth vs. model predictions

Figure 3: Label distributions on the PubMed dataset with M=5.



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

Figure 4: Training curves of different frameworks (GlobSage provides an upper bound).

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 PubMed with five data owners in Fig. 3. For the studied scenario, each data owner only possesses about 20% of the nodes with rather biased label distributions as shown in Fig. 3 (a). Such bias is due the way we synthesize the distributed subgraph system with Louvain clustering, which is also realistic in real scenarios, which essentially makes it hard for any local data owner with limited training samples to obtain a generalized classifier that is globally useful. Although with 13.9% of the links missing across data owners, both FedSage and FedSage+ empower the local data owners in predicting labels that closely follow the ground-true global label distribution as shown in Fig. 3 (b). It is evidently clear from the figure that our FL models exhibit their advantages in learning a more realistic label distribution as our goal in Eq. (1), which is consistent with the observed classification accuracy in Table 2 and our theoretical implications in Section 6.

For the studied case, we also visualize testing accuracy, loss convergence, and runtime along 500 epochs in obtaining $\mathcal{C}$ with FedSage, FedSage+ and GlobSage. The results are presented in Figure 4. Both FedSage and FedSage+ can consistently achieve convergence with rapidly improved testing accuracy. Regarding runtime, even though we conjecture the training of NeighGen to potentially incur significantly more communications and computations, FedSage+ does not consume observable more training time compared to FedSage, since the training data scale of NeighGen is indeed much smaller than GraphSage discussed in Section 4.3.

To sum up, our experimental results quantify the effectiveness of subgraph federated learning, *i.e.*, FedSage and FedSage+, in retrieving a general applicable node classifier in the distributed subgraph systems essentially through involving more training samples from the underlying global graph without directly sharing training data across subgraphs. FedSage+, resolving the unique challenge of missing cross-subgraph links in this setting, further improves FedSage with convincing testing results in all tested scenarios.

6 Implications on Generalization Bound

In this section, we provide a theoretical implication for the generalization error associated with number of training samples, *i.e.*, nodes in the distributed subgraph system. Thus, we are motivated to promote the FedSage and FedSage+ algorithms that include more nodes in the global graph through

collaborative training with FL. To narrow the gap between the real and theoretical settings, we follow Graph Neural Tangent Kernel (GNTK) [28] on universal graph neural networks.

Setting. Our explanation builds on a generalized setting, where we assume a GNN f_{GNN} with layer-wise aggregation operations and fully-connected layers with ReLU activation functions, which includes GraphSage as a special case. The weights of f_{GNN}, W, is i.i.d. sampled from a multivariate Gaussian distribution $\mathcal{N}(0, I)$. For Graph $G = \{V, E\}$, we define the kernel matrix of two nodes $u, v \in V$ as follows.

Definition 6.1 (Informal version of GNTK on node classification) *Considering training an over-parameterized GNN f_{GNN} by gradient descent with infinitesimally small learning rate. Given n training samples of nodes with corresponding labels, we denote $\mathbf{\Theta} \in \mathbb{R}^{n\times n}$ as the the kernel matrix of GNTK. $\mathbf{\Theta}_{uv}$ is defined as*

$$\Theta(u, v) = \mathbb{E}_{W\sim\mathcal{N}(0,I)}\left[\left\langle \frac{\partial f_{\mathrm{GNN}}(W, G, u)}{\partial W}, \frac{f_{\mathrm{GNN}}(W, G, v)}{\partial W}\right\rangle\right] \in \mathbb{R}.$$

Full expression of Θ is shown in the Appendix. The generalization ability in the NTK regime depends on the kernel matrix $\mathbf{\Theta}$. We present the generalization bound associated with the number of training samples n in Theorem 6.2.

Theorem 6.2 (Generalization bound) *Given n training samples of nodes $(u_i, y_i)_{i=1}^n$ drawn i.i.d from the global graph G, consider any loss function $l : \mathbb{R} \times \mathbb{R} \mapsto [0, 1]$ that is 1-Lipschitz in the first argument such that $l(y, y) = 0$. With probability at least $1 - \sigma$ and constant $c \in (0, 1)$, the generalization error of GNTK for node classification can be upper-bounded by*

$$L_{\mathcal{D}(f_{\mathrm{GNTK}})} = \mathbb{E}_{(u',y)\sim G}[l(f_{\mathrm{GNTK}}(G, u'), y)] \lesssim O(1/n^c).$$

Following the generalization bound analysis in [28], we use a standard generalization bound of kernel methods of [29], which shows the upper bound of our GNTK formation error depends on that of $\mathbf{y}^\top \mathbf{\Theta}^{(-1)}\mathbf{y}$ and $\mathrm{tr}(\mathbf{\Theta})$, where $\mathbf{y}$ is the label vector. For the full version of the proofs, please see the Appendix.

Implications. We show the error bound of GNTK on node classification corresponding to the number of training samples. Under the assumptions in Definition 6.1, our theoretical result indicates that more training samples bring down the generalization error [1], which provides plausible support for our goal of building a globally useful classifier through FL in Eq. (1). Such implications are also consistent with our experimental findings in Figure 3 where our FedSage and FedSage+ models can learn more generalizable classifiers that follow the label distributions of the global graph through involving more training nodes across different subgraphs.

7 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, which is consistent with our theoretical implications. 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 inevitable communication cost and potential adversarial analysis during interactions. As communications are vital for collaborative learning, leveraging model compression methods to improve communication efficiency and cryptologic techniques to minimize the privacy leakage risk in the distributed subgraph system can both be promising future directions.

[1] We conjecture the statement can be extended in FL using the techniques in FL-NTK [30].

References

[1] Saif Ur Rehman, Asmat Ullah Khan, and Simon Fong. Graph mining: A survey of graph mining techniques. In *ICDIM*, 2012.

[2] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 2020.

[3] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, 2017.

[4] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[5] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37:50–60, 2020.

[6] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.

[7] Quande Liu, Cheng Chen, Jing Qin, Qi Dou, and Pheng-Ann Heng. Feddg: Federated domain generalization on medical image segmentation via episodic learning in continuous frequency space. *arXiv preprint arXiv:2103.06030*, 2021.

[8] Qi Dou, Tiffany Y So, Meirui Jiang, Quande Liu, Varut Vardhanabhuti, Georgios Kaissis, Zeju Li, Weixin Si, Heather HC Lee, Kevin Yu, 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.

[9] Xinle Liang, Yang Liu, Tianjian Chen, Ming Liu, and Qiang Yang. Federated transfer reinforcement learning for autonomous driving. *arXiv preprint arXiv:1910.06001*, 2019.

[10] Xinghua Zhu, Jianzong Wang, Zhenhou Hong, and Jing Xiao. Empirical studies of institutional federated learning for natural language processing. In *EMNLP*, pages 625–634, 2020.

[11] Chaoyang He, Shen Li, Mahdi Soltanolkotabi, and Salman Avestimehr. Pipetransformer: Automated elastic pipelining for distributed training of transformers. *arXiv preprint arXiv:2102.03161*, 2021.

[12] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, 2017.

[13] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial attack on graph structured data. In *ICML*, 2018.

[14] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *SIGKDD*, 2018.

[15] Daniel Zügner and Stephan Günnemann. Adversarial attacks on graph neural networks via meta learning. In *ICLR*, 2019.

[16] Daniel Zügner and Stephan Günnemann. Certifiable robustness and robust training for graph convolutional networks. In *SIGKDD*, 2019.

[17] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. Graph structure learning for robust graph neural networks. In *SIGKDD*, 2020.

[18] Dingyuan Zhu, Ziwei Zhang, Peng Cui, and Wenwu Zhu. Robust graph convolutional networks against adversarial attacks. In *SIGKDD*, 2019.

[19] Yizhou Zhang, Guojie Song, Lun Du, Shuwen Yang, and Yilun Jin. DANE: domain adaptive network embedding. In *IJCAI*, 2019.

[20] Man Wu, Shirui Pan, Chuan Zhou, Xiaojun Chang, and Xingquan Zhu. Unsupervised domain adaptive graph convolutional networks. In *WWW*, 2020.

[21] Chaoyang He, Keshav Balasubramanian, Emir Ceyani, Yu Rong, Peilin Zhao, Junzhou Huang, Murali Annavaram, and Salman Avestimehr. Fedgraphnn: A federated learning system and benchmark for graph neural networks. *arXiv preprint arXiv:2104.07145*, 2021.

[22] Chuhan Wu, Fangzhao Wu, Yang Cao, Yongfeng Huang, and Xing Xie. Fedgnn: Federated graph neural network for privacy-preserving recommendation. *CoRR*, abs/2102.04925, 2021.

[23] Ross Girshick. Fast r-cnn. In *ICCV*, 2015.

[24] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[25] Galileo Namata, Ben London, Lise Getoor, Bert Huang, and UMD EDU. Query-driven active surveying for collective classification. In *10th International Workshop on Mining and Learning with Graphs*, volume 8, 2012.

[26] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

[27] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[28] Simon S. Du, Kangcheng Hou, Ruslan Salakhutdinov, Barnabás Póczos, Ruosong Wang, and Keyulu Xu. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. In *NeurIPS*, 2019.

[29] Peter L Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3:463–482, 2002.

[30] Baihe Huang, Xiaoxiao Li, Zhao Song, and Xin Yang. Fl-ntk: A neural tangent kernel-based framework for federated learning convergence analysis. *arXiv*, 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See Section 3,4 and 5
   (b) Did you describe the limitations of your work? [Yes] See Section 7.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 7.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Section 6 and Appendix
   (b) Did you include complete proofs of all theoretical results? [Yes] See Section 6 and Appendix
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See Section 5 and the supplemental material
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 5.1
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Table 2

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 5.1

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See Section 5.
   (b) Did you mention the license of the assets? [No] They are all public.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We include our models and code in the supplemental material
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]