# One Node Per User: Node-Level Federated Learning for Graph Neural Networks

**Anonymous Author(s)**
Affiliation
Address
email

## Abstract

Graph Neural Networks (GNNs) training often necessitates gathering raw user data on a central server, which raises significant privacy concerns. Federated learning emerges as a solution, enabling collaborative model training without users directly sharing their raw data. However, integrating federated learning with GNNs presents unique challenges, especially when a client represents a graph node and holds merely a single feature vector. In this paper, we propose a novel framework for node-level federated graph learning. Specifically, we decouple the message-passing and feature vector transformation processes of the first GNN layer, allowing them to be executed separately on the user devices and the cloud server. Moreover, we introduce a graph Laplacian term based on the feature vector's latent representation to regulate the user-side model updates. The experiment results on multiple datasets show that our approach achieves better performance compared with baselines.

## 1 Introduction

Graph Neural Networks (GNNs) have attracted significant attention both within academic circles and across diverse industries. Their remarkable achievements span a multitude of domains, including fraud detection in social networks [32], cancer classification in biology science [20], and materials design in molecular chemistry [4]. In real-world applications, graph data tied to individuals or human behaviors often contains sensitive details. For example, the user's comments, friend list, and profiles on a social platform, as well as their purchase records, browsing history, and transactions on an economic network, are typically deemed private. With increasing emphasis on user privacy, legal restrictions like the General Data Protection Regulation (GDPR) in Europe (EU) and the Health Insurance Portability and Accountability Act (HIPAA) in the US have rendered data-sharing practices infeasible. However, the conventional graph machine learning paradigm requires uploading raw user data to a central server. This is infeasible due to privacy restrictions, hindering the deployment of many real-world graph-based applications.

Federated learning (FL) [16] offers a collaborative learning method, allowing multiple clients to train a model without revealing their raw training data. The workflow of FL follows an iterative training procedure, including multiple communication rounds between the clients and a central server. Specifically, the central server maintains a global model and orchestrates the training process. In each round, the selected clients fetch the global model and perform several epochs of updating using their local training data. The updated local model is later uploaded to the server to produce the latest global model. The training process terminated until the model meets the pre-defined criteria.

Some efforts have recently been devoted to training GNN models over graph data with the preservation of user privacy. One straightforward approach is extending FL to graph machine learning. Depending on the available data possessed by the clients, these research studies [30, 10, 38, 11, 34, 2] assume

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

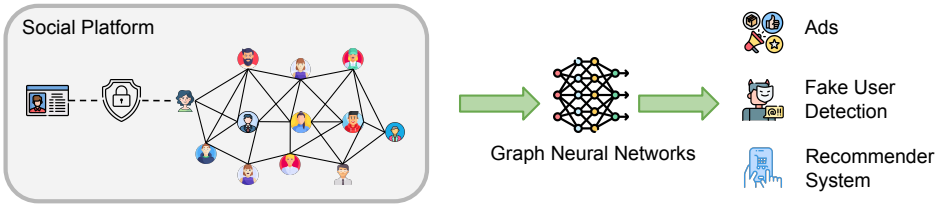



Figure 1: Node-level collaborative training framework for GNNs: a motivating example.

either graph-level FL, where multiple graphs are distributed among clients, or subgraph-level FL, where a big graph is partitioned into several subgraphs and each client has access to the subgraph. With at least one (sub)-graph possessed by the client, local training could proceed as standard FL algorithm plus certain adaptations (e.g., missing link retrieve) tailored for graph data. Note these methods fail under node-level data availability, where each client only possesses data of one node locally (one feature vector). Another line of research work [22] aims to mitigate the privacy risk by introducing Differential-Private (DP) noise to user data, and the model training process follows the conventional machine learning paradigm. As the added DP noise inevitably degrades the quality of training data, this approach suffers from an inherent privacy-performance trade-off. Consequently, the potential of the GNNs to achieve better performance is restricted. In summary, fewer prior works (see Appendix A for detailed related work) have explored training GNNs under node-level user data privacy protection.

**Motivating Example.** As depicted in Fig. 1, our motivating example showcases how industries often use graphs wherein each node represents an individual user. Take mobile social networks as an instance: here, a node represents a mobile user, and an edge signifies the social ties between them. These platforms are interested in using additional user data in user's mobile devices, such as users' locations, activity records, and app usage histories, to perform tasks such as Sybil detection [25], online advertisements [15], and recommendations of social media content [7]. Similarly, telecom giants such as AT&T and Spectrum possess vast amounts of phone call logs between users. These logs can be depicted as a graph where each node symbolizes a user, and an edge indicates a call record shared between two users. To enhance user experience, these companies might desire access to richer user datasets, such as users' app usage statistics, geolocation data, and device sensor information. Such details are instrumental in understanding user preferences and needs.

**Challenges.** Implementing node-level FL within GNNs presents several unique challenges. Foremost, each user's training data is remarkably limited, consisting merely of one feature vector. This is different from existing FL settings, where each client has enough data (a graph or sub-graph) to train a local model. Moreover, the graph-based node classification task belongs to the transductive learning (semi-supervised learning) regime, wherein the unlabeled nodes also contribute to the model training. However, conventional FL algorithms such as FedAvg [16] inherently assume user data to be labeled (or partially labeled). It is untrivial to integrate the knowledge from unlabeled nodes (clients) into the GNNs under the node-level FL setting.

In this paper, we introduce a node-level FL framework for GNNs, termed **nFedGNN**. This framework facilitates collaborative training across multiple clients, with each client only possessing data of a single node[1] in the graph, specifically, one feature vector. Importantly, this feature vector remains localized throughout the training, ensuring no external disclosure. Moreover, we assume that the server has labeled some nodes, and the learning objective is to assign the label for the unlabeled nodes. We also assume the central server has access to the graph topology. Note the research community usually believes the graph topology is private in graph-level and subgraph-level FL. However, under node-level data availability of the client, it is unnecessary to consider the privacy of the graph topology. As illustrated in the motivating example, this assumption aligns with many real-world GNN applications.

**Key ideas of our solution.** To facilitate GNN training across distributed users, our method involves splitting the GNN model between the clients and the server, similar to the technique used in split learning or vertical federated learning [26, 29, 3]. Specifically, the user-side model processes the local

[1]We will use client, user, and node interchangeably in this paper when it does not introduce ambiguity.

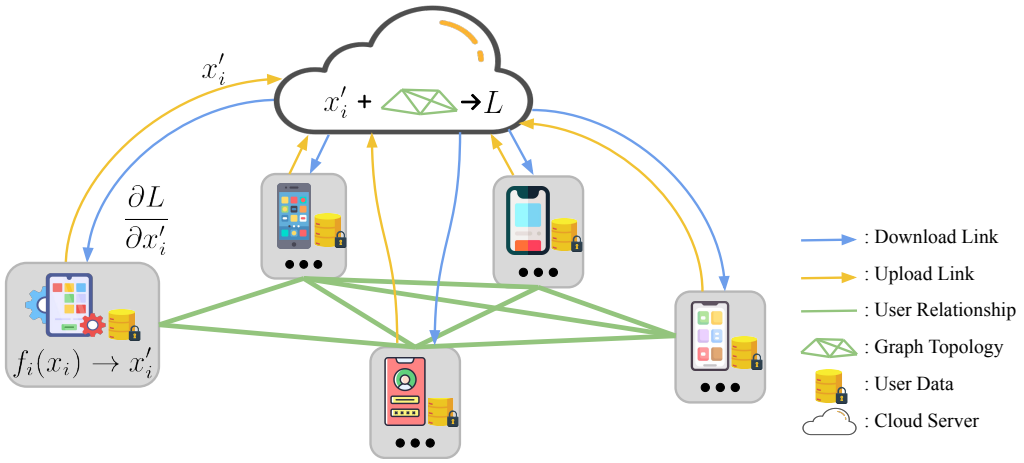



Figure 2: An illustration of the proposed node-level FL framework nFedGNN. Here, each node of the graph is a client of the FL system. The server knows the topology of the graph and the label of a small subset of nodes. The server has no access to the user's raw data while it wants to train a GNN model to assign the label for unlabeled nodes.

private feature vector, yielding a latent representation vector. Subsequently, the server gathers these latent vectors and integrates them with graph topology through the server-side model. To update the client-side model, the server computes the gradient of the latent representation vectors and transmits it back to the clients. The entire training process needs multiple communication rounds until the model meets the predefined criteria. We conducted a preliminary experiment, and the result shows this approach is prone to overfitting, leading to subpar generalization performance.

We analyze the potential causes and conclude that the user-side model of labeled nodes easily fits their local data. To address this, we introduce an explicit graph regularization loss $L_{\text{reg}}$ based on the received latent vectors. Our solution is motivated by the fundamental assumption of graph-based semi-supervised learning: connected nodes in a graph are likely to share the same label [40, 33]. This implies that connected nodes are likely to share a similar node embedding. In the preliminary experiment, this assumption is broken as each user-side model has enough flexibility in its embedding. The regularization loss imposes constraints on the distribution of latent representations, effectively limiting the degree of freedom in the latent space. Consequently, the user-side model is not only required to fit its own local data but also has a strong motivation to learn from its neighbors. As a result, the learned model achieves better performance. We conducted the experiment on several datasets using split GCN and GAT models. The experiment result shows that nFedGNN significantly surpasses the baselines. The contributions of this paper are summarized as follows:

- We propose a node-level FL framework for training GNNs, where each participating user only has access to the data of a single node. The framework inherits the privacy advantage of FL and enables efficient utilization of distributed graph data. Our approach is one of the very few early works in this field, and we are the first to provide a general FL-based approach in this setting.
- We instantiate two popular GNN models, GCN and GAT, within our proposed framework. For both models, the first layer is divided into the server and the users. The remaining layers reside on the server side, functioning as the conventional GCN and GAT models do. Our framework allows us to train the cutting-edge GNN model and achieve better performance without collecting the user's data.
- We perform extensive experiments to evaluate the proposed framework on multiple datasets. The results show that the proposed method outperforms the baselines on all datasets for both GCN and GAT models.

The remainder of this paper is organized as follows. In Section 2, we present details of the GNN and FL system. In Section 3, we describe the proposed algorithm nFedGNN. Then, in Section 4, we

demonstrate experiment results and analysis. Finally, we conclude the paper and discuss the future direction in Section 5.

# 2 Preliminary

## 2.1 Graph Neural Networks

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, X)$ be an attributed graph, where $\mathcal{V}$ is set of nodes, and $\mathcal{E}$ is set of edges. The graph topology can be represented as an adjacency matrix $A \in \mathbb{R}^{n \times n}$, $A_{i,j} = 1$ if there is an edge $(i, j) \in \mathcal{E}$ between node $i$ and node $j$, and $A_{i,j} = 0$ otherwise. Here $n = |\mathcal{V}|$ denotes the number of nodes. $X \in \mathbb{R}^{n \times d}$ is the feature matrix, where each row corresponds to a node's feature vector $x_i \in \mathbb{R}^d$. Denote $Y \in \mathbb{R}^{n \times c}$ as a matrix of the one-hot labels, where each row corresponds to a one-hot vector of a labeled node and $c$ is the number of classes. Note only a subset of the nodes $\mathcal{V}_0 \subset \mathcal{V}$ are labeled.

Given a graph dataset $\mathcal{G}$, the $K$-layer GNN consists of $K$ consecutive layers, where each layer $k$ receives as input the node embeddings $\{\boldsymbol{h}_i^{k-1}, \forall i \in \mathcal{V}\}$ from layer $k-1$ and outputs a new node embedding $\boldsymbol{h}_i^k$ for each node $i$ by aggregating the current embeddings of its adjacent neighbors followed by a learnable transformation as follows:

$$\begin{aligned} \boldsymbol{h}_{\mathcal{N}_i}^k &:= \mathbf{Agg}_k\left(\{\boldsymbol{h}_j^{k-1}, \forall j \in \mathcal{N}_i\}\right), \\ \boldsymbol{h}_i^k &:= \mathbf{Update}_k\left(\boldsymbol{h}_{\mathcal{N}_i}^k\right), \end{aligned} \tag{1}$$

where $\mathcal{N}_i$ is the set of neighbors of node $i$, including itself, in the graph $\mathcal{G}$. Here $\mathbf{Agg}_k(\cdot)$ is the aggregator function (e.g., mean, sum, and max) for $k$-th layer, and $\mathbf{Update}_k(\cdot)$ is the $k$-th layer trainable non-linear function (e.g., neural network) for $k$-th layer. The initial embedding of each node $i$ is its feature vector, i.e., $\boldsymbol{h}_i^0 = \boldsymbol{x}_i$, and the node embeddings from the last layer's output $\{\boldsymbol{h}_i^K, \forall i \in \mathcal{V}\}$ will be used for downstream tasks such as predicting the labels of nodes from the unlabeled set $\mathcal{V} \setminus \mathcal{V}_0$.

## 2.2 Problem Definition

In this paper, we explore learning a GNN model under the node-level federated graph learning setting, where each node in the graph represents a user, and the node feature $x_i$ is private to user $i$. Furthermore, we assume a central server has access to the graph topology $A$ as well as the label of the node in $\mathcal{V}_0$, but can not observe the feature matrix $X$. The central server aims to cooperate with the users to train a GNN model over the graph $\mathcal{G}$ without requiring the private data $x_i$ to leave the users. We focus on the node classification task, where the server wants to assign labels to the remaining unlabeled nodes (users) $\mathcal{V} \setminus \mathcal{V}_0$ in the graph.

## 2.3 FL Objective

Let $f_s(\theta^s, A, \cdot)$ be the server-side model, parameterized by $\theta^s$, and $f_u(\theta_i^u, \cdot)$ be the user-side model, parameterized by $\theta_i^u$. The user-side model's output is $\bar{x}_i = f_u(\theta_i^u, x_i)$. Then, the GNN model's final output is:

$$\begin{aligned} Z &= f_s(\theta^s, A, [\bar{x}_1, \cdots, \bar{x}_n]^T) \\ &= f_s(\theta^s, A, [f_u(\theta_1^u, x_1), \cdots, f_u(\theta_n^u, x_n)]^T). \end{aligned} \tag{2}$$

Define $\theta = \{\theta^s\} \bigcup \{\theta_i^u\}_{i=1}^n$ be the set of all trainable parameters on both user-side and server-side. Then, the goal of GNN training under node-level FL is to minimize the following classification loss:

$$\min_{\theta} \mathcal{L}_{\text{CE}}(\theta) := \sum_{l \in \mathcal{V}_0} \text{CE}(Z_l(X, \theta), Y_l). \tag{3}$$

Here, CE denotes the cross-entropy loss, and $\mathcal{V}_0$ is the set of labeled nodes.

# 3 The Proposed Method

## 3.1 Algorithm to address (3).

**Algorithm 1** nFedGNN

1: **Input:** training data $X = [x_1, x_2, \cdots, x_n]^T$
2: **Output:** learned model $\theta = \{\theta^s\} \bigcup \{\theta_i^u\}_{i=1}^n$
3: **for** round $t = 1, 2, \ldots, T$ **do**
4:  **for** user $i = 1, 2, \ldots, n$, **in parallel do**
5:  Compute the output of user-side model: $x_i' = f_u(\theta_i^u, x_i)$.
6:  Upload $x_i'$ to the server.
7:  **end for**
8:  Server updates $\theta^s$ through gradient decent.
9:  Server computes $\frac{\partial \mathcal{L}_{CE}}{\partial \bar{x}_i}$ and send it back to user.
10:  **for** each user $i = 1, 2, \ldots, n$, **in parallel do**
11:  Compute the gradient for $\theta_i^u$ using $\frac{\partial \mathcal{L}}{\partial x_i'}$.
12:  Update $\theta_i^u$ through gradient decent.
13:  **end for**
14: **end for**

We summarize the details for addressing (3) in **Algorithm 1**. Specifically, at the beginning of the $t$-th communication round, every user (node) in the graph computes the latent output of the user-side model and uploads $x_i'$ to the server (lines 5-6). Subsequently, the server computes the classification loss based on equations (2) and (3). Following this, the server-side model parameter $\theta^s$ can be updated using standard back-propagation and gradient descent (lines 8). The server then computes the gradient for the user-side model's output $\frac{\partial \mathcal{L}_{CE}}{\partial \bar{x}_i}$ and sends it to the corresponding user (line 9). The user receives $\frac{\partial \mathcal{L}_{CE}}{\partial \bar{x}_i}$ and computes the gradient of the user-side model through back-propagation (line 11). Finally, the user-side model parameter $\theta_i^u$ is updated via gradient descent (line 12).

### 3.2 Model Splitting Strategy

Implementing a layer-wise split of the GNN model is unfeasible in a node-level FL setting as the resultant user-side model still requires the entire graph as input. To address this problem, we start from the message-passing mechanism of the GNN layer and suggest splitting the **Agg**$(\cdot)$ and **Update**$(\cdot)$ functions of the first GNN layer. A detailed discussion of the general model splitting process can be found in Appendix B. In this section, we elucidate this concept through a simplified example of a two-layer GCN model.

Let $\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ be the normalized adjacency matrix, where $\tilde{A} = A + I$ is the adjacency matrix with self-connections, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ is the degree matrix, and $I \in \mathbb{R}^{n \times n}$ is the identity matrix. Then, the forward pass of the standard 2-layer GCN model [14] can be expressed as follows

$$Z = \text{softmax}\ (\tilde{A}\ \text{ReLU}(\tilde{A}XW^{(0)})W^{(1)}), \tag{4}$$

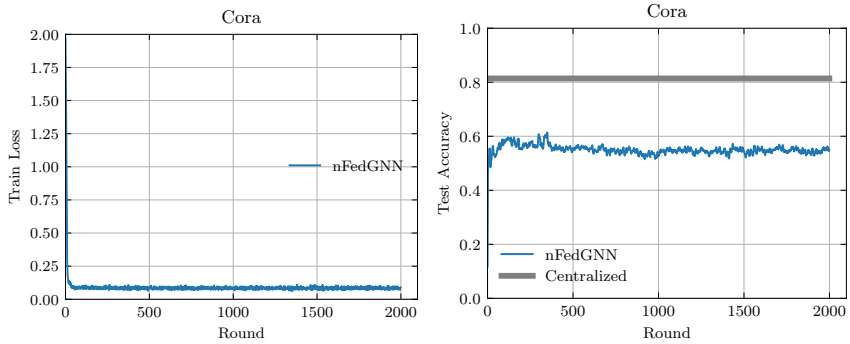
where $W^{(0)}$ and $W^{(1)}$ are the learnable weight matrix of the first and second layers, respectively.

Following the proposed model splitting strategy, we split the first GCN layer into two parts: every user retains a local $W^{(1)}$, while $\tilde{A}$ is located on the server side. The training process of the split GCN model unfolds as follows. In each round, the user-side model first maps its feature vector $x_i$ into latent representation $\bar{x}_i = xW_i^{(0)}$, where $W_i^{(0)}$ is the weight matrix of the user $i$. Subsequently, $x_i$ is uploaded to the server. The server-side model computes the loss based on the topology and the received latent vectors. Let $\bar{X} = [\bar{x}_1, \ldots, \bar{x}_n]^T$. The forward pass of the server-side model is expressed as:

$$Z = \text{softmax}\ (\tilde{A}\text{ReLU}(\tilde{A}\bar{X})W^{(1)}) \tag{5}$$

### 3.3 Preliminary Evaluation

We train the split GCN model on the Cora dataset [24] to assess the performance of the **nFedGNN**. The training curves are shown in Fig.(3). From the figure, we can observe that the training loss quickly decreases after only a few rounds of updating and then does not change much, and improvement in testing accuracy is only seen in the first few rounds. We also plot the testing accuracy of the GCN




(a) (b)

Figure 3: Training curves of a split two-layer GCN on the Cora dataset. Here, we run the experiment with three random seeds and report the average result. The grey line shows the accuracy of the centralized GCN: 81.5 % [14].

in the centralized training setting [14] as reference. Although the proposed method works, the final testing accuracy still leaves room for improvement.

We examine the internal state of the GNN model after splitting. In nFedGNN, we instantiate a local model for every node in the graph. For the user who possesses the labeled node, their local model only undergoes minor adjustments (as the local data is extremely scarce) to fit the local feature vector during training. Consequently, the local model of a labeled user rapidly adapts to their local data, diminishing the training loss to a near-zero value. However, the user-side model from the unlabeled nodes still remains un-updated.

### 3.4 nFedGNN with regularization

To improve the performance of nFedGNN, we revisited the foundational assumption of graph-based semi-supervised learning: the connected nodes in the graph are likely to share the same label [40, 33]. If two nodes are connected in the graph, a properly trained GNN model is likely to assign the same label to them. This suggests that the latent representations of two connected nodes should become increasingly similar as the model's layers become deeper, even if they have distinct feature vectors.

Drawing from this observation, in the context of the split GNN model, if two users are interconnected, their respective latent representations should ideally manifest analogous patterns. However, as discussed previously, the user-side model is only responsible for one feature vector locally. Consequently, the node's latent representation shares less similarity with neighbor nodes. If we restrict the distribution of latent representations and let the connected users have a similar latent representation, the performance of the learned model should be improved. Motivated by this, we introduce the graph Laplacian regularization based on the received latent representations:

$$\mathcal{L}_{\text{reg}} = \frac{1}{\sum_{i=1}^{n} |\mathcal{N}_i|} \sum_{i=1}^{n} \sum_{j=1}^{\mathcal{N}_i} \| f_u(\theta_i^u, x_i) - f_u(\theta_j^u, x_j) \|^2. \tag{6}$$

The overall training loss can be formulated as

$$\mathcal{L} = \mathcal{L}_{CE} + \lambda \mathcal{L}_{\text{reg}}. \tag{7}$$

Here, $\lambda \geq 0$ is the hyperparameter that balances the weight of classification loss and regularization.

### 3.5 Discussion

**Graph Regularization:** Graph regularization-based approach has a long history and is widely used for various applications. It's noteworthy that the state-of-the-art GNN models [14] relax the assumption behind the regularization-based approaches, where the connected nodes in the graph are likely to share the same label. Instead, they directly encode the topology with the feature matrix

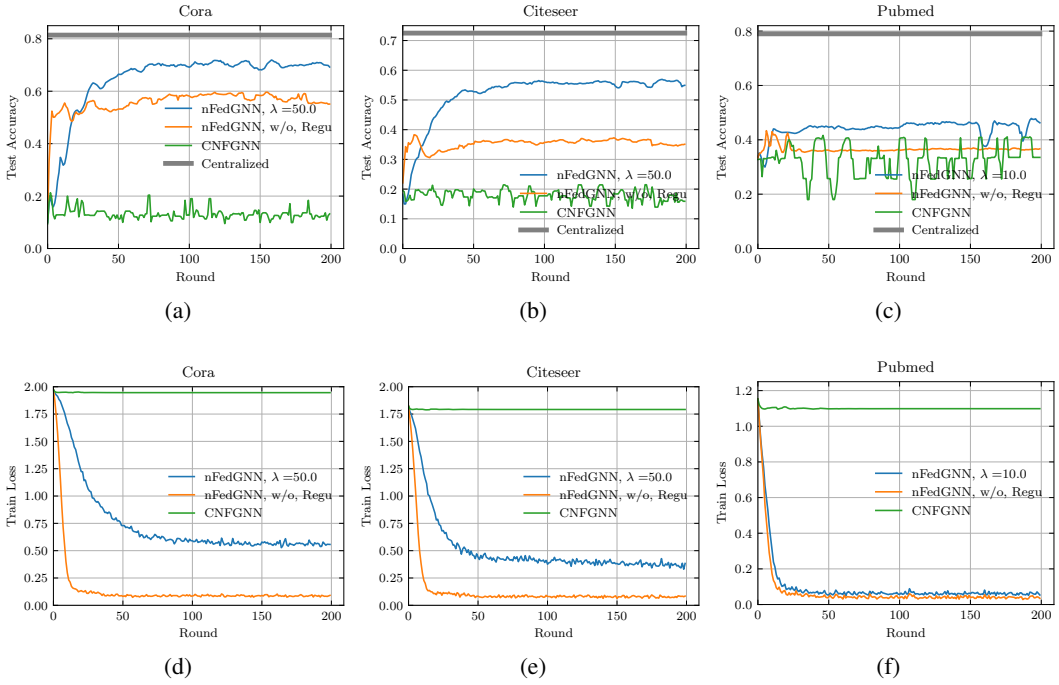



Figure 4: nFedGNN vs baseline in terms of testing accuracy and training loss of a two-layer GCN on the Cora, Citeseer, and Pubmed datasets.

by a neural network, as the graph topology does not necessarily encode similarity but contains the information that does not appear in the feature matrix $X$. Nevertheless, this assumption still holds for a large portion of nodes within the graph, a fact evidenced by the success of regularization-based strategies. In nFedGNN, we use the graph Laplacian regularization to restrict the distribution of latent representation from the user-side model. The remaining layers of the GNN model (located on the server side) still encode the graph topology with the (latent)-feature matrix as the state-of-the-art GNN model. Consequently, the model still utilizes the knowledge from graph topology as state-of-the-art GNNs.

**Communication Cost:** In cross-device federated learning environments, the communication bottleneck is one of the major concerns in prior graph-level and subgraph-level FL researches [38, 13]. However, within nFedGNN, the data exchanged between the user and server each round consists solely of a latent representation vector and a corresponding gradient vector, as opposed to the entire model parameter. Typically, the length of the latent representation vector is in the range of tens to hundreds depending on the tasks and models, substantially smaller compared to the total count of neural network parameters. Thus, within the framework of nFedGNN, communication cost does not pose as a constraining factor.

# 4 Experiments

## 4.1 Experimental Setup

We evaluate nFedGNN on six datasets: Cora, Citesser, Pubmed [24], Chameleon, Squirrel [21], and Wiki-CS [18]. The details and statistics of these datasets can be found in Appendix C. Currently, we assume all users participate in the learning process at every communication round. The algorithm was implemented using DGL [31] with the Pytorch backend, and the training/testing data partition is the same as prior works [14, 28, 19, 18]. All experiments were conducted on a GPU server with 4 NVIDIA RTX A6000 (48GB GPU memory).

We employ two well-established GNN models – GCN[14] and GAT[28] – for the node classification task across all datasets. The model splitting strategy for GCN follows the strategy discussed in

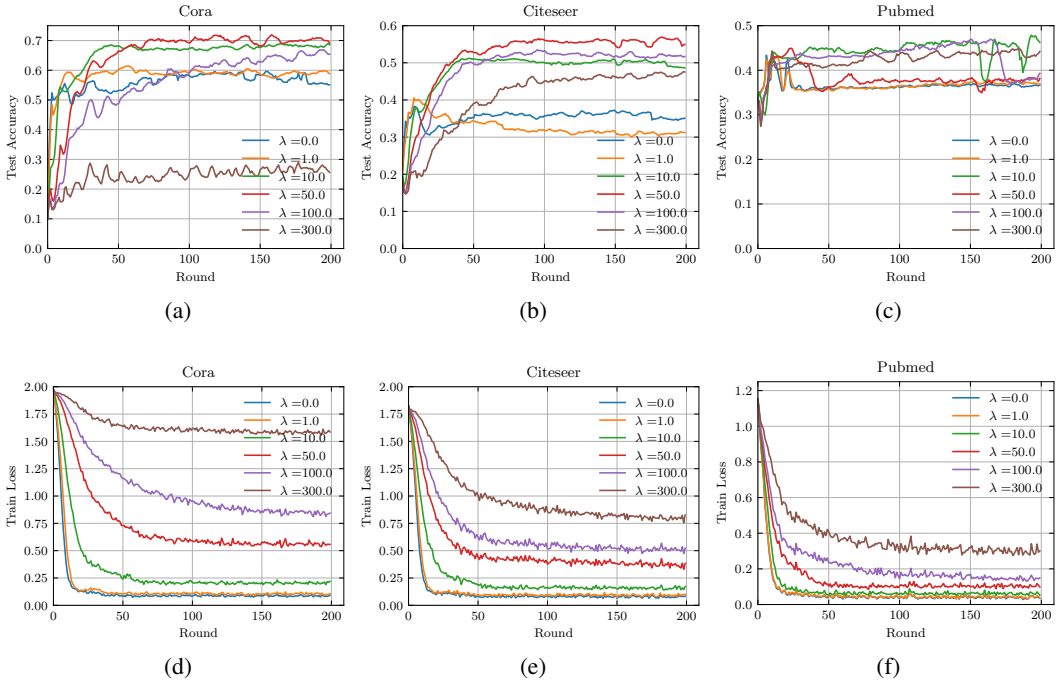



Figure 5: Impact of $\lambda$ of nFedGNN on the Cora, Citeseer, and Pubmed datasets.

Section 2. For GAT, a two-layer model is utilized, and the details of the model splitting strategy are discussed in Appendix D. To ensure a fair comparison, we let the model structure be the same as the model in centralized training counterparts before splitting. Specifically, the GCN model parameters are as follows: 16 hidden neurons, 0.5 dropout rate, and a learning rate of 0.1. For GAT, the dropout rate is 0.6, the number of attention heads is 8, the number of hidden neurons is 16, and the learning rate is 0.01. For both models, we employ the Adam optimizer with weight decay as $5e^{-4}$ to update the model parameter both on the server side and the user side. The total FL round number is set to be 200 for all experiments. We run each experiment with three random seeds and report the averaged training loss and testing accuracy.

**Baseline:** Given limited exploration in prior studies regarding node-level federated graph learning, establishing a baseline for comparison is challenging. A study close to ours is **CNFGNN** [17], which advocates for the transmission of both model parameters and latent representations between the user and the server. However, their focus is traffic prediction using time-series user data, employing an encoder-decoder architecture in their user-side model to handle sequence data, rendering their model and algorithm incompatible with our setting. Nonetheless, we adapt their conceptual framework by allowing users to upload both the model parameter and the latent vector, with the server subsequently averaging the received local models and broadcasting the updated global model to all users. It's important to note that their approach simultaneously uploads the local model and the latent representation, significantly increasing the risk of privacy leakage. Prior research [5] has shown that adversaries could infer sensitive information from model parameters.

### 4.2 Experiment Results

We first compare our method with the baseline. In Fig. 4, we illustrate the training loss and testing accuracy of the split GCN model on the Cora, Citeseer, and Pubmed datasets. Note additional experiment results on Chameleon, Squirrel, and Wiki-CS datasets can be found in Fig. 6 in Appendix E. From the figure, we can observe that nFedGNN significantly outperforms the CNFGNN on all datasets. To show the effectiveness of regularization loss, we search the best value of $\lambda$ within the range $\{0.1, 1, 10, 100, 300\}$ for Cora, Citeseer, and Pubmed, and the range $\{1, 10, 100, 500\}$ for Chameleon, Squirrel, and Wiki-CS. We select the best $\lambda$ for each dataset, and the optimal hyperparameter can be found in each image. In the figure, the performance improvement of nFedGNN

Table 1: Communication cost of GCN and GAT on different datasets (in MB).

| **Method** | **nFedGNN** | | **CNFGNN** | |
|---|---|---|---|---|
| | GCN | GAT | GCN | GAT |
| Cora | 34.06 | 264.45 | $4.74 \times 10^4$ | $3.79 \times 10^5$ |
| Citeseer | 40.61 | 324.90 | $1.50 \times 10^5$ | $1.20 \times 10^6$ |
| Pubmed | 240.69 | $1.93 \times 10^3$ | $1.21 \times 10^5$ | $9.64 \times 10^5$ |
| Chameleon | 27.80 | 222.36 | $6.47 \times 10^4$ | $5.17 \times 10^5$ |
| Squirrel | 63.49 | 507.91 | $1.33 \times 10^5$ | $1.06 \times 10^6$ |
| Wiki-CS | 142.83 | $1.14 \times 10^3$ | $4.31 \times 10^4$ | $3.44 \times 10^5$ |

with the regularization can be clearly observed. The best accuracy reaches $71.9\%$ for the GCN model on the Cora dataset. Here, we also plot the testing accuracy of the GCN model on these datasets in centralized training [14] as references. Although there is still a gap between nFedGNN and the centralized GNN model in testing accuracy, it is significantly reduced compared with the baseline and the no regularization counterpart. Note our goal is not to achieve the state-of-the-art but to demonstrate the effectiveness of the proposed approach. It is worth noting that CNFGNN needs to transmit the extra local model parameters between the users and server at every FL round, necessitating more communication resources. In Appendix E, we demonstrate the comparison results for the GAT in Fig. 7, 8, similar trends and conclusions can be observed in these figures. In summary, nFedGNN has better performance, clearly proving the advantages of nFedGNN compared with baseline.

Subsequently, we investigate the influence of $\lambda$ on the model convergence. As shown in Figure 5, we display the testing accuracy and training loss under different magnitudes of $\lambda$. A general trend emerging from these figures is the testing accuracy increases when $\lambda$ is getting higher. However, when $\lambda$ exceeds some threshold value, the testing accuracy starts to decrease. For the training loss, a higher $\lambda$, the value is larger. Another observation is the convergence speed of testing accuracy is slower with higher $\lambda$, e.g., it takes more rounds for the testing accuracy to achieve the plateau. We conclude that with a higher value of $\lambda$, the training task is more "difficult", and training requires more time to converge. By properly determining the value of $\lambda$, the learned models' performance can be improved. In Appendix E, we also illustrate the impact of $\lambda$ on model convergence for the GAT in Fig. 9. Similar observations can be found in these figures. Note that due to the stochastic nature of the learning process, a few curves may not match the above observations. However, the general trend is clear, and the conclusions are reliable.

### 4.3 Communication Cost Comparison

We count the total communication cost (in MB) of both nFedGNN and CNFGNN for the GCN model and the GAT models in Table 1. This includes the cumulative communication cost spanning 200 training rounds for all users within the graph, with the data transmitted in float32 format. For nFedGNN, we record the size of the latent feature vector. For CNFGNN, we record the size of the latent feature vector and the number of user-side model parameters. In all cases, nFedGNN consumes less communication resources than CNFGNN. Moreover, the total transmitted data size is very small, verifying the conclusion that communication cost is not the bottleneck for our method.

## 5 Conclusion

We introduce nFedGNN, a novel federated learning algorithm for GNNs where each user only has access to the data of a single node. Our method provides an opportunity to utilize the distributed graph data without compromising user privacy. We compare nFedGNN with nFedGNN over multiple datasets and demonstrate the advantage of our method. Despite our extensive efforts, we recognize the limitations in the scope of our study. For example, we have not explored the training on larger datasets and inductive frameworks like GraphSage [9]. In the future, we will extend nFedGNN to adopt partial user participation and to provide rigorous privacy protection (e.g., differential privacy).

# References


[1] Jinheon Baek et al. "Personalized Subgraph Federated Learning". In: *arXiv preprint arXiv:2206.10206* (2022).

[2] Chaochao Chen et al. "Vertically federated graph neural network for privacy-preserving node classification". In: *arXiv preprint arXiv:2005.11903* (2020).

[3] Tianyi Chen et al. "Vafl: a method of vertical asynchronous federated learning". In: *arXiv preprint arXiv:2007.06081* (2020).

[4] David K Duvenaud et al. "Convolutional networks on graphs for learning molecular fingerprints". In: *Advances in neural information processing systems* 28 (2015).

[5] Jonas Geiping et al. "Inverting gradients-how easy is it to break privacy in federated learning?" In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 16937–16947.

[6] Justin Gilmer et al. "Neural message passing for quantum chemistry". In: *International conference on machine learning*. PMLR. 2017, pp. 1263–1272.

[7] Zhiwei Guo and Heng Wang. "A deep graph neural network-based mechanism for social recommendations". In: *IEEE Transactions on Industrial Informatics* 17.4 (2020), pp. 2776–2783.

[8] Otkrist Gupta and Ramesh Raskar. "Distributed learning of deep neural network over multiple agents". In: *Journal of Network and Computer Applications* 116 (2018), pp. 1–8.

[9] William L Hamilton, Rex Ying, and Jure Leskovec. "Inductive representation learning on large graphs". In: *arXiv preprint arXiv:1706.02216* (2017).

[10] Chaoyang He et al. "Fedgraphnn: A federated learning system and benchmark for graph neural networks". In: *arXiv preprint arXiv:2104.07145* (2021).

[11] Kai Hu et al. "FedGCN: Federated Learning-Based Graph Convolutional Networks for Non-Euclidean Spatial Data". In: *Mathematics* 10.6 (2022), p. 1000.

[12] Peter Kairouz et al. "Advances and open problems in federated learning". In: *arXiv preprint arXiv:1912.04977* (2019).

[13] Peter Kairouz et al. "Advances and open problems in federated learning". In: *Foundations and Trends® in Machine Learning* 14.1–2 (2021), pp. 1–210.

[14] Thomas N Kipf and Max Welling. "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907* (2016).

[15] Zekun Li et al. "Fi-gnn: Modeling feature interactions via graph neural networks for ctr prediction". In: *Proceedings of the 28th ACM international conference on information and knowledge management*. 2019, pp. 539–548.

[16] Brendan McMahan et al. "Communication-efficient learning of deep networks from decentralized data". In: *Artificial Intelligence and Statistics*. PMLR. 2017, pp. 1273–1282.

[17] Chuizheng Meng, Sirisha Rambhatla, and Yan Liu. "Cross-node federated graph neural network for spatio-temporal data modeling". In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2021, pp. 1202–1211.

[18] Péter Mernyei and Cătălina Cangea. "Wiki-cs: A wikipedia-based benchmark for graph neural networks". In: *arXiv preprint arXiv:2007.02901* (2020).

[19] Hongbin Pei et al. "Geom-GCN: Geometric Graph Convolutional Networks". In: *International Conference on Learning Representations*. 2019.

[20] Sungmin Rhee, Seokjun Seo, and Sun Kim. "Hybrid approach of relation network and localized graph convolutional filtering for breast cancer subtype classification". In: *arXiv preprint arXiv:1711.05859* (2017).

[21] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. "Multi-scale attributed node embedding". In: *Journal of Complex Networks* 9.2 (2021), cnab014.

[22] Sina Sajadmanesh and Daniel Gatica-Perez. "Locally private graph neural networks". In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2021, pp. 2130–2145.

[23] Simone Scardapane, Indro Spinelli, and Paolo Di Lorenzo. "Distributed Training of Graph Convolutional Networks". In: *IEEE Transactions on Signal and Information Processing over Networks* 7 (2021), pp. 87–100. DOI: 10.1109/TSIPN.2020.3046237.

[24] Prithviraj Sen et al. "Collective classification in network data". In: *AI magazine* 29.3 (2008), pp. 93–93.

[25] Kai Shu et al. “Fake news detection on social media: A data mining perspective”. In: *ACM SIGKDD explorations newsletter* 19.1 (2017), pp. 22–36.

[26] Abhishek Singh et al. “Detailed comparison of communication efficiency of split learning and federated learning”. In: *arXiv preprint arXiv:1909.09145* (2019).

[27] Chandra Thapa, Mahawaga Arachchige Pathum Chamikara, and Seyit Camtepe. “Splitfed: When federated learning meets split learning”. In: *arXiv preprint arXiv:2004.12088* (2020).

[28] Petar Veličković et al. “Graph attention networks”. In: *arXiv preprint arXiv:1710.10903* (2017).

[29] Praneeth Vepakomma et al. “Split learning for health: Distributed deep learning without sharing raw patient data”. In: *arXiv preprint arXiv:1812.00564* (2018).

[30] Binghui Wang et al. “GraphFL: A Federated Learning Framework for Semi-Supervised Node Classification on Graphs”. In: *arXiv preprint arXiv:2012.04187* (2020).

[31] Minjie Wang et al. “Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks”. In: *arXiv preprint arXiv:1909.01315* (2019).

[32] Mark Weber et al. “Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics”. In: *arXiv preprint arXiv:1908.02591* (2019).

[33] Jason Weston, Frédéric Ratle, and Ronan Collobert. “Deep learning via semi-supervised embedding”. In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 1168–1175.

[34] Chuhan Wu et al. “Fedgnn: Federated graph neural network for privacy-preserving recommendation”. In: *arXiv preprint arXiv:2102.04925* (2021).

[35] Yuncheng Wu et al. “Privacy preserving vertical federated learning for tree-based models”. In: *arXiv preprint arXiv:2008.06170* (2020).

[36] Zhaomin Wu, Qinbin Li, and Bingsheng He. “A Coupled Design of Exploiting Record Similarity for Practical Vertical Federated Learning”. In: *Advances in Neural Information Processing Systems*. Ed. by Alice H. Oh et al. 2022. URL: https://openreview.net/forum?id=fiBnhdazkyx.

[37] Han Xie et al. “Federated graph classification over non-iid graphs”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 18839–18852.

[38] Yuhang Yao et al. “Fedgcn: Convergence and communication tradeoffs in federated training of graph convolutional networks”. In: *arXiv preprint arXiv:2201.12433* (2022).

[39] Ke Zhang et al. “Subgraph federated learning with missing neighbor generation”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 6671–6682.

[40] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. “Semi-supervised learning using gaussian fields and harmonic functions”. In: *Proceedings of the 20th International conference on Machine learning (ICML-03)*. 2003, pp. 912–919.

# Appendix

## A Related Work

**Federated Learning.** FL has drawn great attention recently due to its benefits in privacy and communication efficiency [27]. The goal of FL is to allow multiple users to collaboratively train a model under the orchestration of a central server without sharing their raw data [12]. In conventional FL algorithms such as FedAvg [16], model training takes place on the user side, and only the model or model update is transferred between the user and a server. An aggregator on the server subsequently aggregates the models from the users for the next training round. In FL, the user's raw data is never shared.

One branch of FL is vertical federated learning (VFL) [36, 35]. In conventional FL [16], the user has different data samples, while the samples from different users share the same feature space. In VFL, the user owns a common sample space, but disparate feature spaces. The first few layers of the model are partitioned width-wise to accommodate different feature spaces. Each user owns a portion of the whole model corresponding to their feature spaces. The user who possesses the labels holds the remaining layers of the model and coordinates the training process.

Another relevant notion of FL is split learning [8, 29]. Specifically, the model is partitioned layer-wise between the users and the server. Only the intermediate results, e.g., the activation maps and related gradients, are exchanged between the user and server during training. As opposed to FL, split learning requires users to communicate with the server every iteration, incurring heavy communication overhead and high training latency.

**GCN in Distributed Setting.** There has been an increasing research interest in training GNN in a distributed setting. Most of the work focuses on either graph-level FL, where small graphs are distributed among multiple parties [10, 37], or subgraph-level FL, where each party holds a sub-graph of the whole large graph [10, 39, 1, 30, 23].

Specifically, Baek et al.[1] propose a personalized sub-graph FL algorithm, which allows each user to train a personalized model by selectively sharing knowledge across users. Zhang et al. [39] trains a missing neighbor generator to handle the missing edges of the subgraphs across users. Wang et al. [30] employs the model-agnostic meta-learning (MAML) approach to tackle the Non-IID distributed data for the graph-based semi-supervised node classification task. Hu et al. [11] suggested an online adjustable attention mechanism to aggregate the local models. These prior works mainly focus on the setting that each user holds a graph/sub-graph, where a local GNN model could be independently trained and serves as a base model for FedAvg.

In this paper, we consider a more challenging case where each user represents only one node in the graph and does not have access to other's data. It is, in fact, a special but the most challenging case of subgraph-level FL and could not be addressed by any of the prior work. The closest work to ours is [17]. They consider the spatiotemporal dynamic modeling tasks and propose an approach that explicitly encodes the underlying graph structure using GNNs. Their approach is specifically crafted to fit the problem and may be unsuitable for general GNN tasks.

## B General Model Splitting Strategy

Generally, GNN can be described within the Message Passing Neural Networks (MPNN) framework [6]. As shown in (1), the hidden state of each node $\boldsymbol{h}_i^k$ relies on the message from their neighboring nodes. Thus, user $i$ requires $\boldsymbol{h}_{\mathcal{N}_i}^k$ from its neighbor users to compute $\boldsymbol{h}_i^k$. This leads to excessive privacy leakage. Therefore, layer-wise splitting for GNN is infeasible under the node-level FL setting. Alternatively, we seek to split the first layer of the GNN model. However, the conventional GNN layer follows the message-passing-then-encoding protocol. Directly splitting a GNN layer results in the message aggregation function $\mathbf{Agg}_0(\cdot)$ located on the user side. To deal with this problem, we

Table 2: Dataset statistics

| Dataset | Nodes | Edges | Classes | Features | Label Rate |
|---|---|---|---|---|---|
| Cora | 2,708 | 10,556 | 7 | 1,433 | 5.2 % |
| Citeseer | 3,327 | 9,228 | 6 | 3,703 | 3.6 % |
| Pubmed | 19,717 | 88,651 | 3 | 500 | 0.3 % |
| Wiki-CS | 11,701 | 431,726 | 10 | 300 | 5 % |
| Chameleon | 2277 | 36101 | 5 | 300 | 50.0 % |
| Squirrel | 5201 | 217073 | 5 | 2089 | 50.0 % |

reformulated the GNN layer as an encoding-then-message-passing process:

$$\begin{aligned} \bar{\boldsymbol{h}}_i^k &:= \mathbf{Update}_k\left(\boldsymbol{h}_i^k\right), \\ \boldsymbol{h}_i^{k+1} &:= \mathbf{Agg}_k\left(\{\bar{\boldsymbol{h}}_j^k, \forall j \in \mathcal{N}_i\}\right), \end{aligned} \tag{8}$$

Note (8) usually results in the same model as message-passing-then-encoding. Then we can split the first GNN layer into two parts: the update function $\mathbf{Update}_0(\cdot)$ on the user side and the message aggregation function $\mathbf{Agg}_0(\cdot)$ on the server side. Taking the split GCN model 5 as an example, we can find the first layer's hidden state update function $\mathbf{Update}_k(\cdot)$ is a linear projection with parameter $W^{(0)}$, and the message aggregation function $\mathbf{Agg}_k(\cdot)$ is the matrix multiplication with normalized adjacency matrix $\tilde{A}$. Switching the order of the $\mathbf{Update}_k(\cdot)$ and $\mathbf{Agg}_k(\cdot)$ implies the different matrix multiplication orders. The associativity property of matrix multiplication immediately proves the same GCN model after reformulation.

# C Dataset statistics

Cora, Citeseer, and Pubmed serve as citation graphs in which nodes symbolize scientific publications and edges denote citation relationships. Wiki-CS is a dataset grounded in Wikipedia, where nodes represent articles related to Computer Science, and edges signify hyperlinks between them. Similarly, Squirrel and Chameleon are networks comprised of page-to-page connections where nodes are articles from English Wikipedia, and edges depict mutual links between these articles. A summary detailing the number of nodes, edges, and labels for these datasets is provided in Table 2.

# D GAT Splitting Strategy

A GAT layer comprises a learnable weight matrix $W \in \mathbb{R}^{F' \times F}$ and a shared attention mechanism $a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \to \mathbb{R}$, where $F$ and $F'$ are the number of input and output of the GAT layer. Note that $a$ is used to compute the *attention coefficient* $e_{ij}$, which further depends on the feature vectors of nodes $i$ and $j$. Therefore, the user-side model contains the trainable weight matrix $W$, while the shared attention mechanism $a$ should be located on the server side. At each training round, the user-side model first transforms the local feature vector $x_i$ into $x_i' = W_i x_i$, which is uploaded to the server later. The server uses $x_i'$ to compute the attention coefficients $e_{ij} = a(x_i', x_j')$ and update the model. Moreover, the GAT model usually employs the multi-head attention mechanism to stabilize the training process. To achieve this, we let the user have multiple weight matrices. Each weight matrix transforms the input feature independently: $x_{i,l}' = W_i^l x_i'$, where $W_i^l$ is the weight of $l$-th attention mechanism on $i$-th user. Subsequently, the user uploads a set of latent features $\{x_{i,l}'\}_{l=1}^L$ to the server, where $L$ is the number of attention mechanisms. Finally, the output of the first GAT layer is

$$x_i'' = \Big\Vert_{l=1}^{L} \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij}^l x_{i,l}'\right), \tag{9}$$

where $\sigma$ is the activation function and $\alpha_{ij}^l = \text{softmax}(e_{ij}^l)$ is the normalized attention coefficients computed by the $l$-th attention mechanism.

The second GAT layer is the same as that in the GAT model for the task on the Cora dataset in [28]. For the multi-head GAT model, the regularization term of nFedGNN has two variations. The first

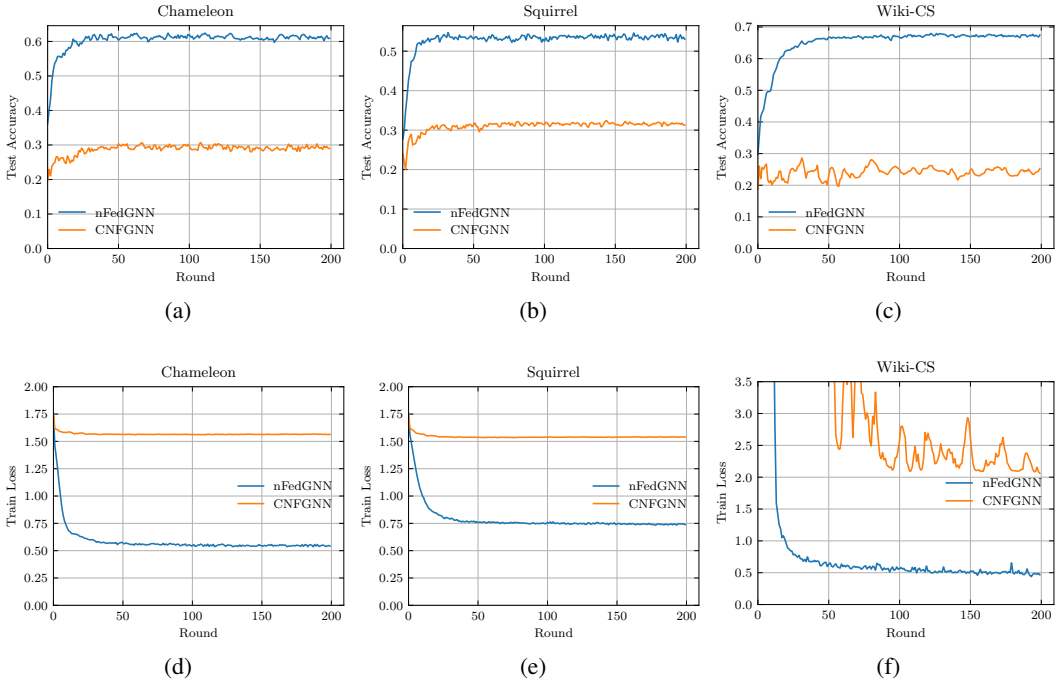



Figure 6: nFedGNN vs CNFGNN for GCN on the Chameleon, Squirrel, and Wiki-CS datasets.

approach treats every element in $\{x'_{i,l}\}_{l=1}^{L}$ equally and computes the regularization for each $x'_{i,l}$ independently. Alternatively, we first compute the average latent representation $\bar{x}'_i = \frac{1}{L}\sum_{l=1}^{L} x'_{i,l}$, and then obtain the regularization term based on $\bar{x}'_i$. In this work, we adopt the first scheme. The performance of the second scheme and the comparison between the two variations are beyond the scope of this work, and we left it for future exploration.

# E Extra Experiment Result

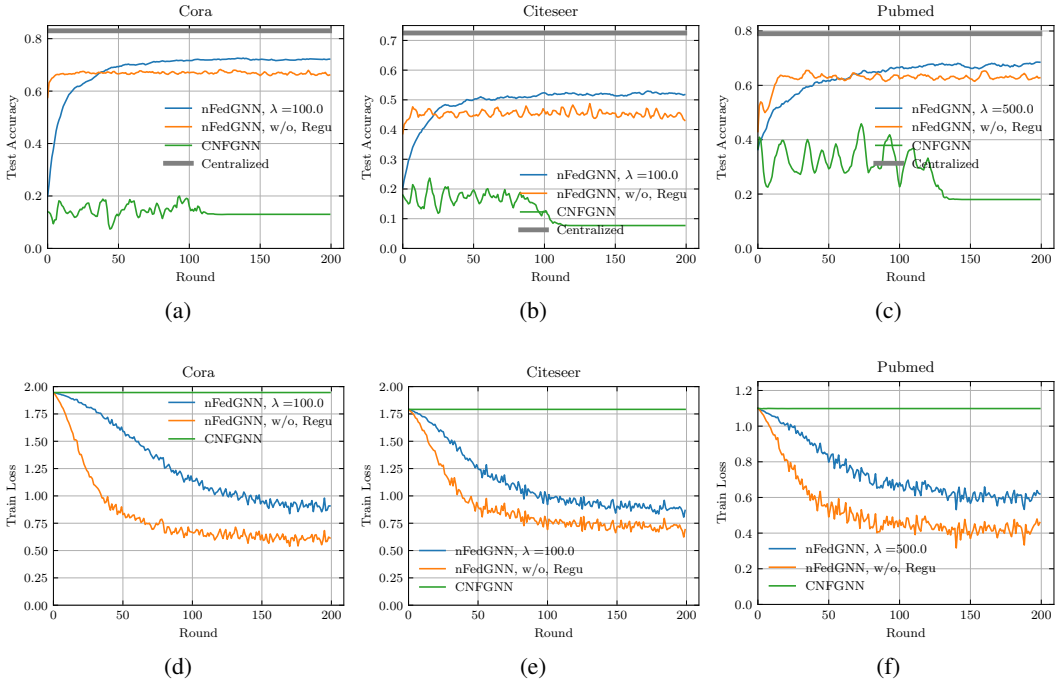



Figure 7: nFedGNN vs CNFGNN for GAT on the Cora, Citeseer, and Pubmed datasets.

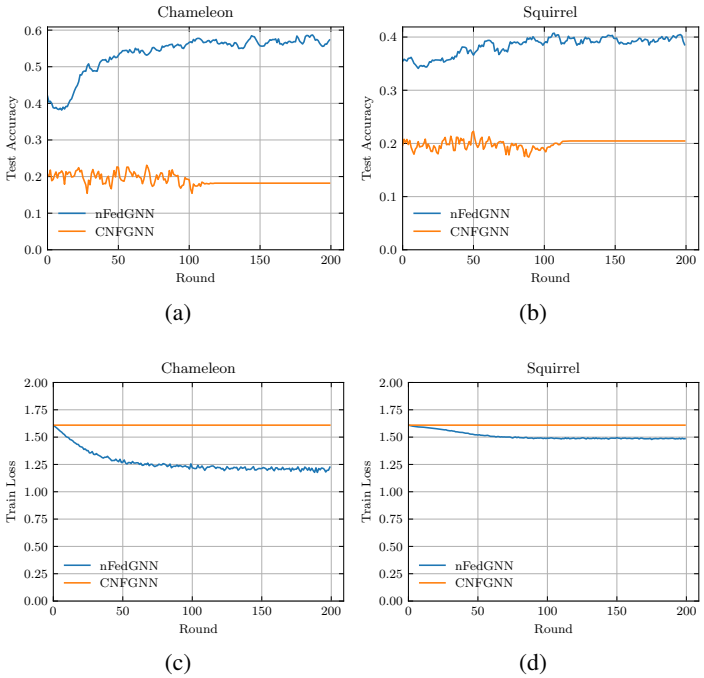



Figure 8: nFedGNN vs CNFGNN for GAT on the Chameleon and Squirrel datasets.

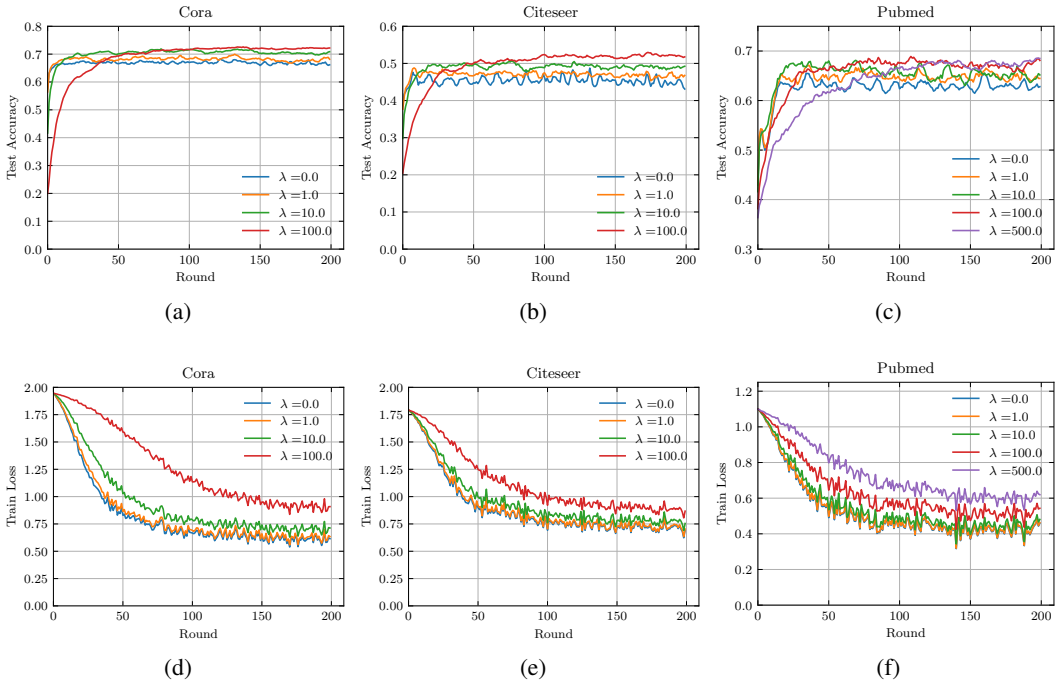



Figure 9: Impact of $\lambda$ for GAT on the Cora, Citeseer, and Pubmed datasets.