# FedEgo: Privacy-preserving Personalized Federated Graph Learning with Ego-graphs

TAOLIN ZHANG, CHENGYUAN MAI, YAOMIN CHANG, CHUAN CHEN, LIN SHU, and ZIBIN ZHENG, Sun Yat-sen University, China

As special information carriers containing both structure and feature information, graphs are widely used in graph mining, e.g., Graph Neural Networks (GNNs). However, graph data are stored separately in multiple distributed parties in some practical scenarios, which may not be directly shared due to conflicts of interest. Hence, federated graph neural networks are proposed to address such data silo issues while preserving each party's privacy (or client). Nevertheless, different graph data distributions of various parties, which is known as the statistical heterogeneity, may degrade the performance of naive federated learning algorithms like FedAvg. In this article, we propose FedEgo, a federated graph learning framework based on ego-graphs to tackle the challenges above, in which each client will train their local models while also contributing to the training of a global model. FedEgo applies GraphSAGE over ego-graphs to make full use of the structure information and utilizes Mixup for privacy concerns. To deal with the statistical heterogeneity, we integrate personalization into learning and propose an adaptive mixing coefficient strategy that enables clients to achieve their optimal personalization. Extensive experimental results and in-depth analysis demonstrate the effectiveness of FedEgo.

CCS Concepts: • **Computing methodologies** → **Neural networks**; • **Information systems** → **Data mining;**

Additional Key Words and Phrases: Ego-graphs, graph neural network, personalized federated learning

**ACM Reference format:**
Taolin Zhang, Chengyuan Mai, Yaomin Chang, Chuan Chen, Lin Shu, and Zibin Zheng. 2023. FedEgo: Privacy-preserving Personalized Federated Graph Learning with Ego-graphs. *ACM Trans. Knowl. Discov. Data.* 18, 2, Article 40 (November 2023), 27 pages.
https://doi.org/10.1145/3624017

## 1 INTRODUCTION

**Graph Neural Networks (GNNs)** have shown incredible performance in distilling information from graph data and deriving expressive node embedding that facilitates downstream tasks such as node classification and link prediction. Nevertheless, previous GNN works focus on centralized

The research is supported by the National Key R&D Program of China (No. 2020YFB1006001), the National Natural Science Foundation of China (62176269), the Guangzhou Science and Technology Program (2023A04J0314).

Authors' addresses: T. Zhang, C. Mai, Y. Chang, C. Chen (Corresponding author), and L. Shu, Sun Yat-sen University, School of Computer Science and Engineering, Higher Education Mega Center, Panyu District, Guangzhou, China; e-mails: {zhangtlin3, maichy7, changym3}@mail2.sysu.edu.cn, chenchuan@mail.sysu.edu.cn, shulin@mail2.sysu.edu.cn; Z. Zheng, Sun Yat-sen University, School of Software Engineering, Tangjiawan Town, Xiangzhou District, Zhuhai, China; e-mails: zhzibin@mail.sysu.edu.cn.

1556-4681/2023/11-ART40 $15.00
https://doi.org/10.1145/3624017

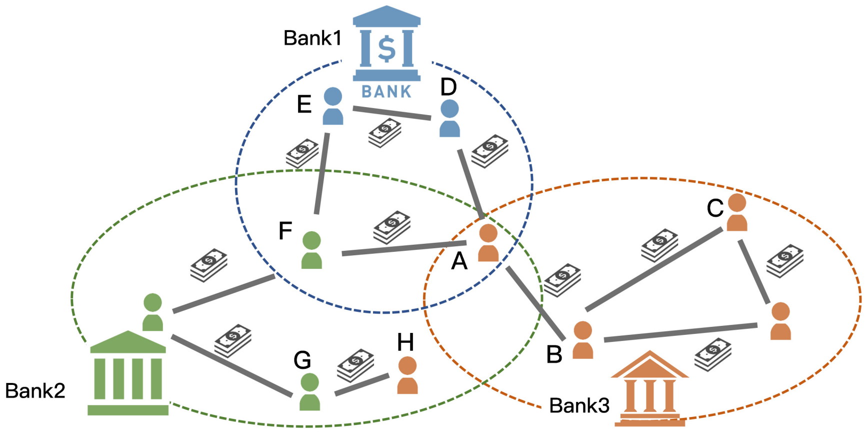


Fig. 1. Motivating scenario in a financial system: Suppose there are three banks in the system, which are marked with different colors. In the system, a graph is formed with customers as nodes and their transactions as edges. People of the same color as the bank represent the bank's target customers who shares the same label, such as wealthy. Customers in different colors hold different labels, and banks desire to distinguish them, which forms a typical node classification task. The dashed lines indicate the detection range of the corresponding bank, some of which may overlap, since someone is likely to use multiple banks. The bank has access to the relevant information about the nodes as well as the transactions between them in the detection range, and most of the nodes are its target customers just as the records of customers A, B, and C storing in Bank 3. However, some of its target customers may only make transfers in other banks without being detected such as customer H. In such realistic system, different target markets give rise to the statistical heterogeneity of the graph data among banks.

node representation learning without taking the data silo issues in the real world into account. In a traditional data silo situation, the data is stored across several distributed parties and it is only allowed to be accessed privately. As a result, solving the problem of how to collaborate on separate graphs from local data owners while preserving privacy for the training of a high-quality graph-based model is crucial.

**Federated learning (FL)**, a technique that decouples the implementation of machine learning from the requirement for direct data sharing, has shown great promise in training models collaboratively while preserving data privacy [14]. The key idea of federated learning is to train a global model in a central server with the contribution of local data owners (or clients). When dealing with graph data, an intuitive idea is to combine naive federated algorithms with graph neural networks directly. Nevertheless, naive FL algorithms based on weight aggregation such as FedAvg do not benefit from the structure information in the graph data and may have poor performance in graph mining. To achieve an effective combination of GNN and FL that we term as federated graph learning, it is essential to fully utilize the structural information of the graph data.

Moreover, federated graph learning suffers from the highly **non-independent identically distributed (non-IID)** problem. Statistical heterogeneity among clients is common in the task of graph mining, which means a single global model might not generalize well on the local data of all clients [13]. As a consequence, personalization needs to be integrated into federated graph learning instead of training a single consensus model. This approach, which involves clients adapting the global model to their own dataset and training their local models for personalization, is known as personalized federated graph learning.

Take a practical problem in the financial system as an example, shown in Figure 1. There are multiple banks in a city's financial system, each storing their records separately due to privacy concerns and conflicts of interest. With customers as nodes and transactions as edges, a local graph could be derived from the records stored in a bank's database. Graph mining could be further applied and one of the most frequently applied tasks may be node classification such as distinguishing different kinds of customers. Furthermore, banks may also collaborate to improve the generalization ability of their local models. In many cases, however, various banks may target different markets and the distributed graph data are in a severe non-IID scenario, making the situation difficult. In other words, the local dataset of a bank can be regarded as a limited observation of the real world, and the key incentive for a bank to participate in collaboration is to reduce its local generalization error with the help of other banks' data. In light of these observations, a bank needs to achieve the tradeoff between the benefit from the collaboration and the disadvantage brought by the potentially statistical heterogeneity, which is a typical case of personalized federated graph learning.

In general, this realistic scenario poses three main challenges in personalized federated graph learning:

— **Challenge 1:** How to make full use of the structure information of graph data during federated training? Since topological information is an indispensable part of graph mining, it is important to integrate it into federated learning organically.
— **Challenge 2:** How to mitigate the issue of the potentially non-IID graph data in the federated learning framework? Observations from different angles of the real dataset lead to severe non-IID graph data and prevent naive FL algorithms from performing well.
— **Challenge 3:** How to achieve an optimal tradeoff between the benefit and the disadvantages of the collaboration? Under the potential non-IID scenario, the ideal situation for a client is to utilize others' data to compensate for its local dataset while minimizing the harm induced by the statistical heterogeneity among each other.

The challenges outlined above motivate us to design FedEgo, a personalized federated graph learning system based on ego-graphs. FedEgo is capable of handling the challenges above and preserves privacy by maintaining the anonymity of the data in terms of both structure and features.

— **To address challenge 1**, it is feasible to view the graph as a family of $k$-hop ego-graphs with structures and node features. Ego-graph, a sampled subgraph with up to k-hop neighbors of the center node, is a kind of useful information carrier and enables message passing of structure and feature information in graph mining. FedEgo extracts topological information from ego-graphs by applying GraphSAGE [10] over them. Importantly, ego-graphs ensure that only local topological information is extracted from the sampled ego-graphs and that the original structure is not recoverable by the server or other clients, which means that they are structure-anonymous.
— **To address challenge 2**, we adopt a strategy of training a global model in the central server to handle the non-IID graph data. Taking consideration of challenge 1, we seek to develop a global model that can effectively capture the structural information contained within ego-graphs. To this end, inspired by References [1, 8], we introduce a network composed of reduction layers and personalization layers to exploit shared low-dimensional embeddings and perform personalized graph mining, respectively. In FedEgo, the clients' model consists of both reduction layers and personalization layers for local training while the global model in the server consists of only the personalization layers to distill information from the ego-graphs in the global dataset. By applying Mixup [27, 28] within each batch in clients, mashed ego-graphs are generated, which are later sent to the server and form the global dataset. In the server, the ability of the global model to deal with non-IID graph data is developed by training over the uploaded mashed ego-graphs. The mashed ego-graphs, which contain only

the mashed embedding and local structure, prevent the transmission of the raw data and protect privacy, and thus they are further feature-anonymous.

— **To address challenge 3**, clients perform updates with the help of the global model. Clients will follow the vanilla algorithm of FedAvg [16] in reduction layers to reach a consensus on encrypting the ego-graphs. They then mix the local and global weights in personalization layers according to an adaptive mixing coefficient. The mixing coefficient, which contributes to achieving better personalization for each client, is adaptively determined by the the difference between the distribution of local and global datasets.

We validate FedEgo on the real-world datasets in non-IID settings to better simulate application scenarios. Experimental results show that FedEgo significantly outperformed baseline methods, thereby fully verifying the effectiveness of FedEgo. We summarized our main contribution as follows:

— We introduce a novel personalized federated graph learning framework based on ego-graphs. FedEgo makes full use of structure information of graph data by applying GraphSAGE over ego-graphs.
— We apply Mixup over ego-graphs for privacy concerns and develop the global model's ability to capture structural information and deal with the non-IID graph data by training over the uploaded mashed ego-graphs.
— We design a strategy to adaptively learn a personalized model for a tradeoff between the benefit and disadvantage of collaboration.
— We conduct extensive experiments on widely used datasets and empirically demonstrate the superior performance of FedEgo under non-IID scenarios.

## 2 RELATED WORK

### 2.1 Federated Learning on Graphs

Recently, federated learning on graphs has raised great interest, and several federated graph frameworks have been proposed by leveraging the power of federated learning and graph neural networks [6, 12, 19, 20, 23, 25, 29]. GraphFL [23] is a model-agnostic meta learning approach designed for few-shot learning, and D-FedGNN [19] is a distributed federated graph framework that enables client collaboration without a centralized server. FedSage+ [29], which trains a missing neighbor generator to recover the missing edges cross clients, mainly targets distributed subgraph systems that are not common in practice. FedGL [6] uploads prediction results and embeddings for global information of nodes, while FedGCN [26] exchanges average information about the node's neighbors among clients. Both of them suffer from severe privacy problems, since the server and others know whether a specific node is in a certain client's local dataset. FedEgo, by contrast, preserves privacy for node classification tasks in a realistic setting by keeping the data anonymous in terms of structure and feature. Recent research DP-FedRec [20] applies differential privacy and utilizes K-hop extensions to obtain extended graphs for federated learning. DP-FedRec extends the existing intersection subgraph by using K-hop neighbor information, as opposed to FedEgo, which constructs a K-hop ego-graph for each node. However, the communication cost of transmitting the extended graphs may be prohibitively high, and the original structure of the local graph is exposed to others through the extended graphs, resulting in privacy risks.

### 2.2 Personalized Federated Learning

Personalization in federated learning has attracted much attention and has been widely explored in recent approaches [1, 9, 11, 13, 15]. Reference [1] proposes a neural network with base layers

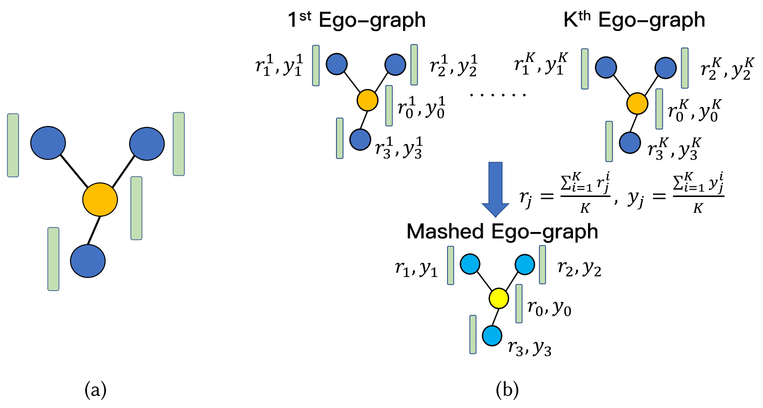


Fig. 2. (a) Illustration of 1 hop ego-graph. (b) Illustration of the alignment and Mixup among a batch of ego-graphs. The center nodes are aligned together and their neighbors are extended recursively. The reduction embedding $r$ and one-hot label $y$ are averaged according to the alignment.

for federated averaging and personalization layers for personalization. Reference [8] is a further extension of Reference [1] and obtains low-dimensional embedding to accelerate convergence. Some other works [7, 13] borrow ideas from MAML, and Reference [13] considers federated learning as an instance of MAML. Moreover, Reference [23] introduces MAML into the federated graph learning framework and designs GraphFL for few-shot learning. Furthermore, methods such as using a mixture of global and local models [9, 15], multi-task learning [22], and adding proximal terms to perform local fine-tuning [11] have been proven effective for improving the performance of personalized models. In contrast to existing work, our approach focuses on achieving better personalization by considering the differences between local and global distributions and providing theoretical justification for the proposed method.

## 3 PRELIMINARIES

### 3.1 Ego-graph

Let $G = (V, E)$ be a graph with $N$ nodes, where $V = \{v_1, \ldots, v_N\}$ denotes the node set and $E \subseteq V \times V$ denotes the edge set. Following References [2, 31], we have the definition of a $k$-hop ego-graph.

*Definition 3.1.* ($k$-hop ego-graph). A graph $g_v = \{V_v, E_v\}$ is called a $k$-hop ego-graph centered at node $v$ if it has a $k$-layer centroid expansion such that the greatest shortest path rooted from $v_i$ has length $k$, i.e., $k = \max_{v_i \in V} |S(v, v_i)|$, where $S(v, v_i)$ is the shortest path from $v$ to $v_i$ and $|\cdot|$ denotes the length of the path.

Given $k$, a depth-based representation of the graph can be described as a family of ego-graphs $G = \{g_{v_i}\}_{i=1}^N$ [2, 31]. The label of each node is retained in the ego-graph, which is represented as a one-hot vector. As useful information carriers, ego-graphs will be sampled in each client for the training in FedEgo. In practice, we follow the method in GraphSAGE [10] and sample a fixed-size set of neighbors in each layer for the nodes. Note that the sampled ego-graphs will be in a fixed shape and can be trained independently of the original graph.

When training with the family of ego-graphs, we do not concern about the concrete information in the original graph but only the local property. In this sense, there is no way of knowing where the center node is located in the original graph and one could not restore the original graph through the sampled ego-graphs. Therefore, the sampled ego-graphs are structure-anonymous.

### 3.2 GraphSAGE

GraphSAGE [10] is a method applicable to inductive graph learning, with which we could enable the information flow in ego-graphs. It focuses on the local property of the target node and aggregates features from the neighborhood to obtain embedding for downstream tasks. Let $h_v^{(l)}$ denote the embedding of node $v$ in the layer $l$, and the convolution layer with a mean aggregator can be defined as:

$$h_v^{(l)} = \sigma(W^{(l)} \cdot h_v^{(l-1)} + W^{(l)} \cdot Mean(\{h_u^{(l-1)}, u \in N(v)\})), \tag{1}$$

with $W^{(l)}$ as the weight matrix and $N(v)$ as the 1-hop neighbors of $v$. In particular, $h_v^{(0)}$ denotes the input features. For supervised node classification tasks, we use cross-entropy as the loss function. Let $(x, y)$ denote the node samples with feature and label, $f_c : X \to S$ is the map function from feature space $X$ to the label space $S$, $\mathbf{1}_{y=c}$ is the indicator function of label $c$, $W$ is the weight in the whole model. Then, the cross-entropy loss could be formulated as:

$$\ell(W) = \mathbb{E}_{x,y\sim P}\left[\sum_{c=1}^{C} \mathbf{1}_{y=c} \log f_c(x, W)\right] = \sum_{c=1}^{C} P(y = c)\mathbb{E}_{x|y=c}[\log f_c(x, W)], \tag{2}$$

where $P$ represents the probability vector of data distribution and $\sum_{c=1}^{C} P(y = c) = 1$. The distribution vector $P$ can be further obtained by following formula:

$$P = \frac{\sum y_{one-hot}}{|N|}, \tag{3}$$

where $y_{one-hot}$ is the one-hot vector of label and $|N|$ is the number of nodes.

### 3.3 Federated Learning

In federated learning, the main objective is to learn a global model by clients' collaboration and a typical implementation is FedAvg [16]. In FedAvg, clients perform local updates and global averages iteratively and finally learn a global model. Formally, we assume that there are $N$ clients with $W_i$ as the local weight of client $i$. In a training epoch, the server first broadcasts the latest global weight $W_g$ to all the clients. Subsequently, clients load the weight $W_i = W_g$ and then perform local updates several times:

$$W_i = W_i - \alpha \nabla L_i(W_i). \tag{4}$$

After that, the server averages the uploaded local parameters and obtains the new global model:

$$W_g = \frac{1}{N} \sum_{i=1}^{N} W_i. \tag{5}$$

The process is repeated for convergence and a global model is learned for prediction. FedAvg aggregates information from models whereas it ignores the difference between clients. Previous works show that it may perform poorly when it comes to severe non-IID situations [25].

### 3.4 Mixup

Mixup [28] is a data augmentation technique that applies linear combination over data samples to generate additional data. In the federated framework, Yoon et al. [27] propose FedMix and apply Mixup over samples within each batch. Formally, given a batch of $n$ data samples $\{(x_i, y_i)\}_{i=1}^{n}$, FedMix constructs augmented data samples by averaging features and labels:

$$\begin{cases} \tilde{x} = \dfrac{\sum_{i=1}^{n} x_i}{n} \\ \tilde{y} = \dfrac{\sum_{i=1}^{n} y_i}{n}. \end{cases} \tag{6}$$

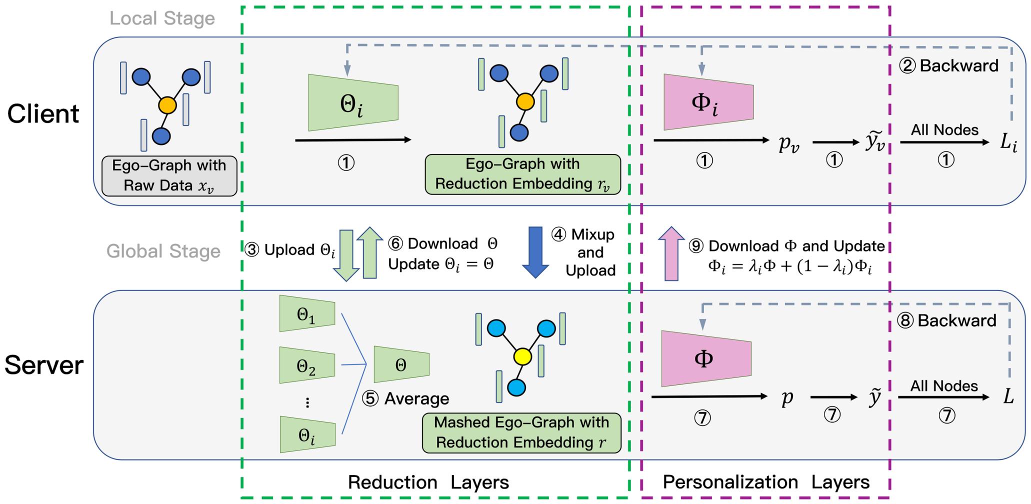


Fig. 3. The detailed framework of FedEgo.

## 4 PROPOSED METHOD: FEDEGO

In FedEgo, we expect the server to be capable of capturing both structural and feature information of non-IID graph data from clients. And then, with the help of the global model, clients update their local model for better personalization. To this end, there are different network architectures between clients and the server. Specifically, a client's local model consists of reduction layers and personalization layers, whereas there are only personalized layers in the server. Reduction layers are designed to exploit shared low-dimensional embeddings among clients, while personalization layers are used for personalized graph mining. To extend GraphSAGE over the ego-graphs, we follow Reference [10] and sample a fixed-size set of neighbors for each given node. And then the training of FedEgo can be mainly divided into local stage and global stage. (1) In the **Local Stage**, clients feed their local ego-graphs into reduction layers and obtain the low-dimensional embedding. Subsequently, Mixup is applied over ego-graphs to generate mashed ego-graphs within each batch. Then, the embedding obtained from reduction layers will be further fed into personalization layers. Eventually, each client calculates the loss and updates the parameters in reduction layers and personalization layers. (2) In the **Global Stage**, clients upload parameters in reduction layers and the mashed ego-graphs to the server for collaboration. The server aggregates the parameters by applying FedAvg algorithm and updates the global personalization layers by training over the mashed ego-graphs. After that, all the parameters are sent back to clients. Clients will then load the reduction layers and update their personalization layers by mixing the local and global weight. The framework of FedEgo is illustrated in Figure 3, and Algorithm 1 demonstrates the training process.

### 4.1 Local Stage

*4.1.1 Reduction Layers: Multi-Layer Perception.* In the federated framework, data distribution between clients is in a severe non-IID situation, which gives rise to the difficulty to train a central model. However, these heterogeneous data may share a common representation despite having various labels [8], and hence reduction layers are proposed to capture the common low-dimensional embeddings across clients, as shown in the green part of Figure 3. And the embedding obtained by reduction layers, which we term reduction embedding, may facilitate subsequent calculations while also protecting data privacy due to the reduction of dimensionality. Besides, it

**ALGORITHM 1:** The algorithm of FedEgo

```
Require: Clients number $N$, node feature matrix $\{X_i\}$, initialized weight in reduction layers and personalization layers of each client $\Theta_i$ and $\Phi_i$, initialized weight in personalization layers of the global model $\Phi_g$
Ensure: Model parameters for each client $\{\Theta_i\}$ and $\{\Phi_i\}$
 1: Clients sample $k$ hop ego-graphs in the fixed shape from their local datasets and calculate local distribution vector $P_i$ by Equation (3).
 2: while not converge do
 3:     Local Stage:
 4:     for client $i = 1$ to $N$ do in parallel
 5:         for each epoch do
 6:             for each batch do
 7:                 for each node $v$ within the batch with feature $x_v \in X_i$ do
 8:                     $r_v = \Theta_i(x_v)$ // Local reduction layers
 9:                     $p_v = \Phi_i(r_v)$ // Local personalization layers
10:                 end
11:                 // Generate a mashed ego-graph
12:                 Calculate the embedding and the label of each node in the mashed ego-graph by Equation (8)
13:                 // Parameters update
14:                 Calculate loss $\ell_i$ by Equation (2)
15:                 Set $\Theta_i \leftarrow \Theta_i - \eta\nabla\ell_i, \Phi_i \leftarrow \Phi_i - \eta\nabla\ell_i,$
16:             end
17:         end
18:     end
19:     Global Stage:
20:     // Averaging in reduction layers
21:     Clients send local weight in reduction layers $\Theta_i$ and Server averages the parameters by $\Theta_g = \frac{\sum_{i=1}^{N}\Theta_i}{N}$
22:     // Training of the global model
23:     Clients send all the mashed ego-graphs with embedding and label to Server
24:     for each epoch do
25:         for each batch do
26:             for each node with mashed embedding $r$ do
27:                 $p = \Phi_g(r)$ // Global personalization layers
28:             end
29:             // Parameters update
30:             Calculate loss $\ell_g$ by Equation (2)
31:             Set $\Phi_g \leftarrow \Phi_g - \eta\nabla\ell_g$
32:         end
33:     end
34:     // Parameters update for clients
35:     Server calculates the gloal distribution vector $P_g$ by Equation (3)
36:     for client $i = 1$ to $N$ do in parallel
37:         Calculate $EMD_i$ and $\lambda_i$ by Equation (14) and (15)
38:         Set $\Theta_i \leftarrow \Theta_g, \Phi_i \leftarrow \lambda_i\Phi_g + (1 - \lambda_i)\Phi_i$
39:     end
40: end
```

has been observed that deeply stacking the layers often results in significantly worse performance for GNNs, which is called over-smoothing [5]. To avoid such a problem, herein, we dig out the reduction embedding using a Multi-Layer Perception, since 2-layer GNN will be applied in personalization layers for graph mining. Formally, consider $x_v \in \mathbb{R}^d$ as the $d$-dimension raw feature of node $v$ and a $l_r$ layers Perception, the hidden embedding can be calculated as follows:

$$r_v^{(l)} = \sigma\left(W_r^{(l)} \cdot r_v^{(l-1)} + b_r^{(l)}\right), \tag{7}$$

where $W_r^{(l)}$ and $b_r^{(l)}$ is the weight and the bias of layer $l$, respectively, $\sigma$ is the activation function, and $r_v^{(l)}$ represents the hidden embedding of node $v$ obtained by layer $l$. Specifically, we take the raw feature as input of the reduction layers and we have $r_v^{(0)} = x_v$. The final embedding $r_v^{(l_r)}$ is regarded as the reduction embedding. For convenience, we term the whole reduction layers as $\Theta$.

*4.1.2 Mashed Ego-graphs: Mixup within Each Batch.* In FedEgo, clients will generate mashed ego-graphs for the training of the global model and further send them to the server rather than raw data. As mentioned above, all mashed ego-graphs uploaded are structure-anonymous, as it only contains the local property rather than the whole graph. Therefore, the server has no way of knowing the structure of the original graphs and whether a specific node exists in the local dataset of a certain client merely according to the local topology. To further protect the privacy of the client, we apply Mixup over the ego-graphs to make the feature indistinguishable to protect the original node.

The mashed ego-graphs are in the same shape as the ego-graphs and can be obtained by mixing up the reduction embedding of ego-graphs within each batch. As a popular data augmentation technique for traditional data, Mixup, however, is not directly applicable to graph data because of the alignment problem. The feature matrices of traditional data samples share the same dimension and can be used for element-wise operations, while the irregularity of structures in graph data limits the element-wise calculation and various ways to mix up may give rise to different impacts on the training.

Mixup, however, is relatively easy to be applied over ego-graphs in the fixed shape. The number of $k$-hop neighbors in such ego-graphs is only determined by $k$ and the size of the neighbors set $n$. For convenience, we sort the nodes based on its layer to assign each node a specific position and obtain the position sets $Q_i$ in the $i$th layers, i.e., $Q_0 = \{0\}, Q_1 = \{1, 2, \ldots n\}, Q_2 = \{n+1, n+2, \ldots n^2\}$, and so on. The node in a specific position connects to its corresponding neighbors in the next layer. For example, the neighbors of node 1 in the second layer can be represented as $\{n+1, n+2, \ldots 2n\}$. The position within the same layer can be assigned arbitrarily as long as the special connectivity between layers is maintained.

To apply Mixup over ego-graphs, we could first align the center nodes, i.e., the node 0 of all the ego-graphs within a batch together and then recursively extend their neighbors to assign each node a specific position in its corresponding layer. Then, the embedding of each position in the mashed ego-graph can be obtained by simply averaging the embedding of the corresponding nodes according to the alignment, which is illustrated in Figure 2(b). Formally, given the embedding and the one-hot label of the $j$th position in the $i$th ego-graphs $r_j^i$ and $y_j^i$ according to the alignment, we obtain the mashed embedding $r_j$ as well as the label $y_j$ of the nodes in the mashed ego-graphs:

$$r_j = \frac{\sum_{i=1}^{n} r_j^i}{n}, y_j = \frac{\sum_{i=1}^{n} y_j^i}{n}. \tag{8}$$

The mashed ego-graphs average the reduction embedding to prevent privacy leakage, making them feature-anonymous in addition to structure-anonymous. In other words, the mashed ego-graphs can be considered to be virtual samples generated from the original distribution and protect the original nodes from being detected by attackers. Besides, there are lots of ways to align ego-graphs, though, all of them have the same effect on a specific GNN model according to Theorem 4.1.

THEOREM 4.1. *Mashed ego-graphs generated from all sorts of alignments are equivalent for the training of linear GraphSAGE layers without activation functions. If there are two different alignments A and B, the final embedding of the center node after aggregation under alignment A and B will be the same.*

The detailed proof of Theorem 4.1 is provided in Appendix A. As Theorem 4.1 shows, linear GraphSAGE layers avoid the impact that different alignments bring, and hence they could be used as the personalization layers without incurring bias. However, the unbiased estimate of the embedding may not always guarantee better performance when dealing with complex scenarios, such as the distribution of highly non-IID data among clients. Due to the high statistical heterogeneity, activation function may be preferable in the personalization layers, since it empowers the model's ability to learn complex patterns while also introducing bias that will alleviate the severe non-IID issues. In developments test, we found a slight improvement of GraphSAGE with activation over the linear one and thus focus on the former for the rest of our experiments.

*4.1.3 Personalization Layers: GraphSAGE over Ego-graphs and Classification.* Once the reduction embedding is obtained, we further feed the ego-graph into linear GraphSAGE layers to forward implement graph mining, as can be seen in the pink part of Figure 3. Formally, we have the personalization layers:

$$p_v^{(l)} = \left( W_p^{(l)} \cdot p_v^{(l-1)} + W_p^{(l)} \cdot \frac{\sum_{u \in N(v)} p_u^{(l-1)}}{|N(v)|} \right), \tag{9}$$

where $p_v^{(l)}$ represents the hidden embedding of node $v$ obtained by layer $l$. And we have $p_v^{(0)} = r_v^{(l_r)}$, i.e., using the reduction embedding as input. Moreover, it is worth noting that we use linear GraphSAGE layers without activation functions between them. After that, predictions are generated by a subsequent linear classifier with a softmax function, and the loss is calculated eventually. For convenience, we term the whole personalization layer as $\Phi$.

## 4.2 Global Stage

*4.2.1 Global Training.* In the global stage, the server trains a global model over the mashed ego-graphs uploaded by clients and updates the parameters in the global model. There exist the same personalization layers in the server as in clients and we have GNN layers defined like Equation (9) with a classifier in the global model:

$$p_v^{(l)} = \sigma\left( W_g^{(l)} \cdot p_u^{(l-1)} + W_g^{(l)} \cdot \frac{\sum_{u \in N(v)} p_u^{(l-1)}}{|N(v)|} \right), \tag{10}$$

where $W_g^{(l)}$ denotes the $l$th layer global personalization layers weight.

When considering the communication cost, another variant of FedEgo is to replace the reduction layers with GNN and the personalization layers with MLP. An incentive in this variant is that clients simply need to upload their reduction embedding of the center nodes instead of the structure in the mashed ego-graphs. However, the information flow of topological structure is implicit in the form of the reduction embedding, making it rather challenging for the server to train, especially in the case of highly non-IID data. Averaging the GNN parameters in the reduction layers may also be inappropriate, since different data distribution among clients contributes to various node connection patterns. Consequently, clients will receive limited or even negative help because of the poor collaboration.

When using the GNN layers as the personalization layers, the generalization ability of the global model is obviously better than any local one because of the access to the encrypted data among all clients. After that, clients perform the following updates to enhance the performance of their local models:

*4.2.2 Averaging in Reduction Layers.* Aiming at exploiting shared representations among clients, the parameters in reduction layers are updated by coordinate-wise weight averaging. Formally, we have the following update in reduction layers:

$$\Theta_g = \frac{1}{N}\sum_{i=1}^{N}\Theta_i, \tag{11}$$

$$\Theta_i = \Theta_g, \tag{12}$$

where $\Theta_i$ represents the reduction layers weights in client $i$ and $\Theta_g$ denotes averaged one. With the update in reduction layers, clients reach a consensus to some degree even if graph data from their datasets are potentially non-IID. In other words, clients encrypt their raw data in the same way and project their data into a low-dimensional space.

*4.2.3 Mixing in Personalization Layers.* When updating parameters in personalization layers, clients will mix local and global weight to achieve the tradeoff of advantages and drawbacks of the global model. Given a mixing coefficient $\lambda_i$ for client $i$, we update the personalization layers as follows:

$$\Phi_i = \lambda_i \cdot \Phi_g + (1-\lambda_i)\cdot\Phi_i, \tag{13}$$

where $\Phi_i$ represents the personalization layers weights in client $i$ and $\Phi_g$ denotes global one. Intuitively, the strategy to select mixing coefficient hinges on the diversity between local and global distributions. With great diversity, the mixing coefficient is expected to be large in that the client tends to gain more from the global model to correct its deviations and reduce the local generalization error. Otherwise, $\lambda$ needs to be small for better personalization of the local model.

*4.2.4 Adaptive Mixing Coefficient for Each Client.* It is obvious that a fixed $\lambda$ is inappropriate for all the clients due to the potential statistical heterogeneity. We further propose an adaptive strategy to select $\lambda$ with theoretical analysis, thereby adjusting $\lambda$ for each client to achieve better personalization. For convenience, we termed the model weights in clients and the server as local and global weight, respectively. First, following Reference [30], a metric is designed in Definition 4.2 to measure the distance between local and global weights. We only analyze the personalization layers here, since the reduction layers are frozen by averaging. Then, we bound the weight divergence in Theorem 4.3.

*Definition 4.2.* Let $\Phi_i$ denote the local weight in personalization layers on client $i$, $\Phi_g$ denote the global one, then the weight divergence $WD_i$ is defined as the distance between local and global weight: $WD_i = \|\frac{\Phi_i-\Phi_g}{\Phi_g}\|$.

Theorem 4.3. *Given $N$ clients with $K$ samples of nodes $(x_k, y_k)_{k=1}^{K}$ drawn i.i.d from local data distribution $P_i$ for client $i \in [N]$ and the global data distribution $P_g$. Let training SGD update steps for clients and server be the same $E_c = E_s = t-1$. Consider the update in personalization layers as a separate step and it is conducted every $t$ steps. The local weight of client $i$ and the global weight in the $t$th step of the $T$th epoch are denoted as $\Phi_{i,t}^{(T)}$ and $\Phi_{g,t}^{(T)}$, respectively. Let $\nabla_w \mathbb{E}_{x|y=c}\log f_c(x,w)$ be $L_{x|y=c}$-Lipschitz for each class $c \in [C]$ and the mixing coefficient $\lambda_i$, then the bound of weight* *divergence after T th update is formulated as follows:*

$$\|\Phi_{i,t}^{(T)} - \Phi_{g,t}^{(T)}\| \leq (1-\lambda_i)a^{t-1}\|\Phi_{i,t}^{(T-1)} - \Phi_{g,t}^{(T-1)}\| + \eta(1-\lambda_i)\sum_{c=1}^{C}\|P_i(y=c) - P_g(y=c)\|\left(\sum_{j=0}^{t-2} a^j g_{\max}(\Phi_{g,t-2-j}^{(T)})\right),$$

*where* $g_{\max}(w) = \max_{c=1}^{C} \|\nabla_\Phi \mathbb{E}_{x|y=c} \log f_c(x,w)\|$ *and* $a = 1 + \eta \max_{c=1}^{C} L_{x|y=c}$.

The proof is provided in Appendix B. The second part on the right side includes the difference between the distribution of local and global datasets that is mainly reflected in $\sum_{c=1}^{C}\|P_i(y=c) - P_g(y=c)\|$, which is termed as **earth mover distance (EMD)**. EMD is correlated with $\lambda_i$ and thus we can fine-tune $\lambda_i$ to affect the impact of EMD and optimize the weight divergence. When the difference of local and global distribution is large (larger $EMD$), a larger $\lambda_i$ should be selected to pull the client closer to the global model. Otherwise, $\lambda_i$ should be small to fully achieve the personalization. Then, we can obtain $EMD_i$ of client $i$:

$$EMD_i = \sum_{c=1}^{C}\|P_i(y=c) - P_g(y=c)\|. \tag{14}$$

We further provide a formula to adaptively select $\lambda$ by introducing a hyperparameter $\gamma$ in Equation (15). It ensures $1-\lambda_i$ to be negatively correlated with $EMD_i$ and $\lambda_i$ varies from 0 to 1 when $EMD_i$ increases:

$$\lambda_i = \left(\frac{EMD_i}{2}\right)^{\gamma}. \tag{15}$$

Both the local distribution vector $P_i$ and the global one $P_g$ can be calculated by Equation (3) even though there are only averaged data in the server. We could pre-compute $P_i$ of each client and update $P_g$ as well as $EMD_i$ in the global stage. The mixing coefficient is dynamically determined by the $EMD_i$ and helps clients to adjust the update of their local models to achieve best personalization performance.

## 4.3 Privacy Risks and Communication Cost Analysis

In light of previous approaches, it is a common practice for clients to share information of its local distribution with each other. In other words, clients should provide more useful knowledge through expressive intermediate results in addition to the shared parameters and gradients. In FedSage [29], the neighbor generator with high accuracy develops each client's ability to predict the node feature from other client's data distribution. In FedMix [27], clients exchange averaged images as additional data and have access to other client's data distribution to some degree. In FedGCN [26], model exchanges the average information about the node's neighbors among clients. Thereby, it is an important concern to determine in what way should the client's local data distribution be presented to others. In FedEgo, clients upload not only the parameters but also the mashed ego-graphs as information carriers, and hence the privacy risks and whether the additional communication cost is affordable are significant issues. A key factor that affects the privacy risks and the communication cost is the batch size.

With a low batch size, more mashed ego-graphs result in a larger communication cost, and more sensitive information of original data will pass over, yielding the problem of privacy leakage. However, in the other extreme, a large batch size leads to a significant amount of information loss and poor performance of the global model. Based on the analysis above, appropriate batch size is exactly a tradeoff of privacy protection and client collaboration. Adding Gaussian noise, which is associated with differential privacy [17], is another typical practice of providing additional privacy.

Table 1. The Statistics and Relevant Setting of Four Datasets

| Dataset | \|V\| | \|E\| | #C | N | $\alpha_{global}$ | $\alpha_{local}$ |
|---|---|---|---|---|---|---|
| Cora | 2,708 | 5,429 | 7 | 5 | 0.3 | 0.3 |
| Citeseer | 3,312 | 4,715 | 6 | 5 | 0.3 | 0.3 |
| Wiki | 17,716 | 52,867 | 4 | 10 | 0.3 | 0.2 |
| CoraFull | 19,793 | 63,421 | 70 | 10 | 0.3 | 0.3 |
| FedDBLP | 52,202 | 271,054 | 4 | 7 | Default Split | Default Split |

|V| and |E| show the number of node and the edges, respectively. #C denotes the number of classes. $N$ indicates the number of clients chosen in the experiments. $\alpha_{global}$ and $\alpha_{local}$ are the sample rate of the global test dataset and local dataset, respectively. $lr$ is the learning rate of the Adam optimizer.

By adding Gaussian noise into the mashed ego-graphs, we find that FedEgo is robust to the noises with a small decline in the performance.

Extensive experiments have been produced to seek a best tradeoff of privacy and communication cost over batch size and prove the robustness of FedEgo.

## 5 EXPERIMENT

### 5.1 Datasets and Experimental Settings

We conduct our experiments on four real datasets: Cora [21], Citeseer [21], CoraFull [4], Wiki [18], and FedDBLP [24]. Table 1 shows the details of the datasets and the relevant settings. In our experiments, we construct a local dataset for each client and a global dataset for the final test, which are used for verifying the personalization and the generalization ability of the model, respectively. We first sample nodes for global dataset and delete them from the original dataset, leaving the remaining nodes for clients' local dataset. To construct a label distribution skew scenario, the nodes are divided into different sets, depending on the label. After that each client randomly selects 3 labels as its major node labels and samples nodes from the corresponding sets to compose 80% nodes in its local dataset. Random unselected nodes will be added as the remaining 20% nodes. In each client, 300 nodes are sampled for testing and 20% nodes for validating, leaving other nodes for training. We choose 2 hop neighbors for each node and set the number of neighbors to be 6 when sampling ego-graphs.

### 5.2 Comparison Methods

We compare FedEgo with the following methods:

- **Local Only:** In this method, each client trains its model by feeding local data independently.
- **FedAvg:** FedAvg [16] applies the averaging method over the weight parameters to obtain a global model. It is a simple but effective way to cope with the non-IID scenario. In this method, the parameters are averaged in both the reductions and the personalization layers.
- **FedProx:** FedProx [14] tackles data heterogeneity by adding a proximal term to the loss. We apply the adaptive FedProx loss coefficient in [0.001,0.01, 0.1,1] based on the fluctuation of the loss.
- **GraphFL:** GraphFL [23] is a method designed for few-shot learning based on **model-agnostic meta-learning (MAML)** and addresses the problem of non-IID graph data between clients. We follow the original setting in Reference [23] and use 100 nodes for both the support and query set in GraphFL.
- **D-FedGNN:** D-FedGNN [19] is a distributed federated graph framework based on the weighted communication topology among clients. We follow the setting in Reference [19] and use the standard ring network for aggregation.

- **FedGCN:** In FedGCN [26], clients communicate with each other to exchange average information about the node's neighbors. It suffers from severe privacy problems, since others know the nodes in a certain client's local dataset. We follow the original setting in Reference [26] and 2 GCN layers for FedGCN.
- **FedSage:** In FedSage [29], clients collaborate and apply the FedAvg over the GraphSAGE framework. FedSage is specifically designed for subgraph federated learning, and we follow the original architectures without implementing the reduction layers and personalization layers.
- **FedSage+:** FedSage+[29] improves upon FedSage by training a neighbor generator on top of it. This addition enhances the model's ability to capture hidden relationships across distributed local graphs.

The models are in the same structure with 1 layer MLP as the reduction layers, 2 layers GraphSAGE with activation function as the personalization layers, followed by 1 fully connected layer as the classifier. We choose Relu as the activation function and Adam as the optimizer. Besides, the amount of the mashed ego-graphs and the additional communication cost are both strongly influenced by batch size, which is preferred to be 32 based on observations in the development test 5.8. During the clients' training, nodes will be trained in 5 mini-batches for 5 epochs each round. In FedEgo, the server will utilize ego-graphs uploaded by clients for training for 5 epochs each round. We execute all experiments 4 times and the averaged results are reported.[1]

## 5.3 Label-skew Scenarios Analysis

*5.3.1 Personalization Ability.* The F1 score in the local test given in Table 2 illustrates the personalization ability of each method under severe non-IID scenarios. It is a clear finding that the result in the local test is much higher compared to the global test, primarily because of the same distribution of the testing and training data. Furthermore, FedEgo, FedAvg, FedProx, and D-FedGNN benefit from the collaboration on all datasets and enhance the personalization ability of local models. FedSage+ and FedSage achieve similar results in the local test, but both are lower than FedEgo most of the time. D-FedGNN performs better than FedAvg, since the average merely with local neighbors mitigates the issues of non-IID to some degree. For FedEgo, there is only a slight performance improvement than FedAvg, which is caused by the label skew scenario and the information from others only compensates for a client's training to a limited extent. Interestingly, GraphFL performs worse than other FL methods in all cases, owing to the fact that it is designed for few-shot learning and does not extract enough useful information under the severe label distribution skew scenario. Similarly, FedGCN is not suitable for non-IID scenarios and is no match for the local training when it comes to enormous data, as the results on Wiki and CoraFull demonstrate.

*5.3.2 Generalization Ability.* As can be seen from the result in Table 3, the most striking observation emerging from the comparison is that FedEgo consistently outperforms other methods and improves the generalization ability of clients' local models. With updates to the reduction and personalization layers, clients achieve approximately an 11%–15% improvement in performance compared to local training. The remarkable improvement indicates that FedEgo can facilitate client collaboration and address non-IID graph data.

FedSage demonstrates noticeable improvements over local training, while FedSage+ gains a little boost further from the neighbor generator, which is consistent with the original results in Reference [29]. Interestingly, FedSage+ experiences out-of-memory errors when training over large datasets such as CoraFull, due to enormous virtual nodes generated by its neighbor

[1]Code available at https://github.com/FedEgo/FedEgo

Table 2. F1 Score for Node Classification in the Local Test under Label-skew Scenarios

| Dataset | Local Only | FedAvg | FedProx | GraphFL | D-FedGNN | FedGCN | FedSage | FedSage+ | FedEgo |
|---|---|---|---|---|---|---|---|---|---|
| Cora | 0.8437 (±0.0039) | 0.9473 (±0.0012) | 0.9483 (±0.0019) | 0.867 (±0.0029) | 0.9503 (±0.0017) | 0.8784 (±0.0006) | 0.9507 (±0.0009) | 0.952 (±0.0008) | **0.9577** (±0.0012) |
| Citeseer | 0.7617 (±0.0005) | 0.918 (±0.0029) | 0.918 (±0.0014) | 0.755 (±0.0014) | 0.9193 (±0.0005) | 0.8967 (±0.0008) | 0.913 (±0.0008) | 0.9137 (±0.0005) | **0.9210** (±0.0024) |
| Wiki | 0.8728 (±0.0141) | 0.9258 (±0.0101) | 0.9232 (±0.0096) | 0.8088 (±0.0069) | 0.92 (±0.0097) | 0.817 (±0.0040) | 0.9223 (±0.0083) | **0.9246** (±0.0075) | 0.9191 (±0.0077) |
| CoraFull | 0.6402 (±0.0002) | 0.874 (±0.0010) | 0.873 (±0.0009) | 0.477 (±0.0017) | 0.8837 (±0.0003) | 0.8466 (±0.0025) | 0.881 (±0.0003) | Out Of Memory | **0.8972** (±0.0008) |

Table 3. F1 Score for Node Classification in the Global Test under Label-skew Scenarios

| Dataset | Local Only | FedAvg | FedProx | GraphFL | D-FedGNN | FedGCN | FedSage | FedSage+ | FedEgo |
|---|---|---|---|---|---|---|---|---|---|
| Cora | 0.6985 (±0.0014) | 0.7706 (±0.0033) | 0.7697 (±0.0037) | 0.7346 (±0.0027) | 0.7865 (±0.0022) | 0.6933 (±0.0007) | 0.7926 (±0.0018) | 0.7848 (±0.0026) | **0.8016** (±0.0019) |
| Citeseer | 0.6125 (±0.0003) | 0.6941 (±0.0058) | 0.6924 (±0.0038) | 0.6327 (±0.0070) | 0.7049 (±0.0055) | 0.6614 (±0.0009) | 0.7055 (±0.0011) | 0.7071 (±0.0012) | **0.7200** (±0.0015) |
| Wiki | 0.696 (±0.0113) | 0.7856 (±0.0020) | 0.7851 (±0.0034) | 0.7112 (±0.0061) | 0.7960 (±0.0014) | 0.4428 (±0.0310) | 0.7839 (±0.0006) | 0.7849 (±0.0001) | **0.8126** (±0.0100) |
| CoraFull | 0.4905 (±0.0006) | 0.5351 (±0.0045) | 0.5336 (±0.0050) | 0.3328 (±0.0032) | 0.5615 (±0.0011) | 0.4777 (±0.0005) | 0.599 (±0.0006) | Out Of Memory | **0.6221** (±0.0006) |

generator. Additionally, FedSage+ generates neighbors with huge feature dimensions identical to the original node, which is not practical or affordable in real-world scenarios due to privacy concerns and memory costs.

Similar to FedProx, FedAvg provides a boost to some amount, whereas it still falls short of FedEgo. Clients benefit from collaboration in the reduction and personalization layers, as evidenced by the gap between FedAvg and local training. Besides, D-FedGNN performs slightly better than FedAvg due to its unique aggregation. Compared to FedAvg, FedEgo used a personalized pattern rather than averaging updates in personalization layers. FedEgo significantly exceeds FedAvg and D-FedGNN, implying that a mixture of the local and global models is far superior to a simple average in the severe non-IID scenario. GraphFL and FedGCN suffer from the same issue mentioned above and perform poorly in all cases.

## 5.4 Community Clustering Scenarios Analysis

Constructing federated graph datasets by splitting the global graph via community clustering is a common practice. Such methods rely on the assumption that nodes within a cluster have a higher degree of interaction than nodes across different clusters. When creating federated graph datasets, community detection algorithms are applied to extract communities from large networks. These clusters are then assigned to the clients to preserve the general graph structure. The Louvain method [3] is a popular clustering method that maximizes the modularity of the split clusters and is well-suited for community detection. In this study, we use the Louvain community_splitter introduced by FedDBLP [24] and leave the nodes in a single cluster as the global test.

*5.4.1 Personalization Ability.* We introduce the FedDBLP dataset in the community clustering scenarios, and the results of it are compatible to other datasets. The fact that the performance of model decreases significantly stands out in the result (Table 4), indicating that the local structure of the global graph captured by the community detection algorithm may not always be easy for GNNs to train and study.

FedEgo still performs outstandingly in the local test and outperforms both FedSage and FedSage+ by a large margin. It is noteworthy that FedGCN shows remarkable results in the local test while

Table 4. F1 Score for Node Classification in the Local Test under Community Clustering Scenarios

| Dataset | Local Only | FedAvg | FedProx | GraphFL | D-FedGNN | FedGCN | FedSage | FedSage+ | FedEgo |
|---|---|---|---|---|---|---|---|---|---|
| Cora | 0.4473 (±0.0005) | 0.7107 (±0.0012) | 0.7083 (±0.0050) | 0.639 (±0.0037) | 0.71 (±0.0014) | **0.7287** (±0.0003) | 0.6963 (±0.0031) | 0.6960 (±0.0014) | **0.7287** (±0.0041) |
| Citeseer | 0.4770 (±0.0051) | 0.6947 (±0.0017) | 0.6953 (±0.0017) | 0.5997 (±0.0019) | 0.702 (±0.0008) | **0.7951** (±0.0003) | 0.677 (±0.0022) | 0.6877 (±0.0012) | 0.7153 (±0.0012) |
| Wiki | 0.7373 (±0.0073) | 0.7986 (±0.0030) | 0.7973 (±0.0020) | 0.6843 (±0.0081) | 0.7999 (±0.0010) | 0.7265 (±0.0010) | 0.7983 (±0.0010) | 0.797 (±0.0023) | **0.8128** (±0.0013) |
| CoraFull | 0.2318 (±0.0089) | 0.3904 (±0.0021) | 0.3924 (±0.0069) | 0.1809 (±0.0088) | 0.5378 (±0.1268) | **0.7202** (±0.0005) | 0.4962 (±0.0023) | Out Of Memory | 0.5113 (±0.0021) |
| FedDBLP | 0.7413 (±0.0018) | 0.7765 (±0.0014) | 0.7773 (±0.0010) | 0.6506 (±0.0038) | 0.7684 (±0.0036) | **0.8383** (±0.0004) | 0.7802 (±0.0006) | 0.7807 (±0.0017) | 0.7851 (±0.0012) |

Table 5. F1 Score for Node Classification in the Global Test under Community Clustering Scenarios

| Dataset | Local Only | FedAvg | FedProx | GraphFL | D-FedGNN | FedGCN | FedSage | FedSage+ | FedEgo |
|---|---|---|---|---|---|---|---|---|---|
| Cora | 0.5492 (±0.0013) | 0.728 (±0.0024) | 0.7275 (±0.0023) | 0.6653 (±0.0042) | 0.7355 (±0.0006) | 0.6229 (±0.0003) | 0.7173 (±0.0017) | 0.7312 (±0.0022) | **0.7536** (±0.0018) |
| Citeseer | 0.5505 (±0.0027) | 0.6738 (±0.0047) | 0.6747 (±0.0026) | 0.6174 (±0.0016) | 0.6847 (±0.0015) | 0.6071 (±0.0016) | 0.6754 (±0.0002) | 0.6755 (±0.0011) | **0.6921** (±0.0023) |
| Wiki | 0.7393 (±0.0068) | 0.8019 (±0.0053) | 0.8038 (±0.0052) | 0.6912 (±0.0062) | 0.8008 (±0.0049) | 0.6704 (±0.0045) | 0.8059 (±0.0031) | 0.8022 (±0.0004) | **0.8143** (±0.0012) |
| CoraFull | 0.2548 (±0.0014) | 0.3924 (±0.0032) | 0.3934 (±0.0008) | 0.1585 (±0.0041) | 0.3794 (±0.0751) | 0.2735 (±0.0002) | 0.4785 (±0.0016) | Out Of Memory | **0.4906** (±0.0001) |
| FedDBLP | 0.6654 (±0.0033) | 0.8124 (±0.0011) | 0.8151 (±0.0018) | 0.8071 (±0.0015) | 0.8172 (±0.0014) | 0.6313 (±0.0004) | 0.8173 (±0.0011) | 0.8204 (±0.0001) | **0.8285** (±0.0005) |

it performs very poorly in the global test and even lower than local training in some cases. We hypothesize that this is because FedGCN is not able to model the connection between nodes across different graphs and thus focuses on the nodes within the local datasets. Therefore, both FedGCN and D-FedGNN lack the generalization ability over the global test and cannot generalize well to the ground truth distribution. And the performance of other methods is consistent with the results obtained from the label-skew scenarios.

*5.4.2 Generalization Ability.* The results in the global test under the community clustering scenarios are shown in Table 5. FedEgo continues to achieve state-of-the-art results in the global test, surpassing FedSage by 1.2%–3.7% and FedSage+ by 0.9%–1.2%. It is worth noting that while FedAvg and FedProx perform comparably to FedSage in some cases, they still fall short of FedSage+. The results of other methods are consistent with the conclusion drawn from the label-skew scenarios.

## 5.5 The Effort of Statistical Heterogeneity

The performance of FedAvg and FedEgo is heavily influenced by the statistical heterogeneity among clients that is controlled by the major node rate. With a larger major node rate, clients tend to have more nodes in the same classes, leading to a higher statistical heterogeneity. According to this, we vary the major node rate and provide the results in Table 6. Unsurprisingly, the generalization ability of both FedEgo and FedAvg reduces as the major node rate becomes larger. Perhaps more surprisingly, the increase in major node rate improves the personalization ability to some level. In this case, more nodes in the local dataset will share the same labels and the pattern of interconnections between nodes becomes relatively constant and easy to learn.

## 5.6 Ablation Study

To verify the effectiveness of Mixup, we undertake a comparison method called *FedEgo w/o Mixup,* wherein clients upload the first ego-graph within a batch to the server rather than the mashed

Table 6. F1 Score on Wiki under Different Major Node Rates

| | Global Test | | | Local Test | | |
|---|---|---|---|---|---|---|
| Major Node Rate | 0.0 | 0.3 | 0.8 | 0.0 | 0.3 | 0.8 |
| FedAvg | 0.803 | 0.800 | 0.785 | **0.907** | 0.906 | 0.922 |
| FedEgo | **0.823** | **0.826** | **0.818** | 0.903 | **0.907** | **0.926** |

Table 7. Comparison between FedEgo, FedEgo-Linear, and FedEgo w/o Mixup

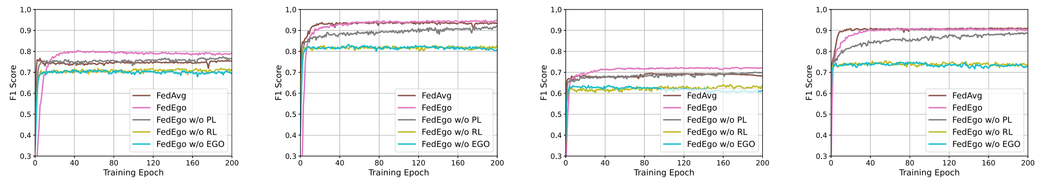
| | Global Test | | Local Test | |
|---|---|---|---|---|
| Methods | Cora | Citeseer | Cora | Citeseer |
| FedEgo w/o Mixup | 0.783 | 0.717 | 0.951 | 0.912 |
| FedEgo-Linear | 0.793 | 0.717 | 0.958 | 0.918 |
| FedEgo | **0.794** | **0.727** | **0.959** | **0.920** |

ego-graphs. Note that it is not feasible to do so in real scenarios, since the transmission of raw ego-graphs may result in the risk of privacy leakage. We also compare FedEgo with its linear variant, denoted as *FedEgo-Linear*, to evaluate the impact of the activation function in the personalization layers. FedEgo-Linear guarantees unbiased estimate of the center nodes' embedding, whereas FedEgo applies activation function to learn more complex patterns in the data.

As the result in Table 7 indicates, Mixup improves both the generalization ability and personalization ability of the model while preserving privacy. The reason is that the graph data among clients are severely non-IID and the virtual samples generated by Mixup can be considered as the clients' exploration of the unknown distribution in the real-world dataset, thereby improving the robustness of the model. By comparing FedEgo with FedEgo w/o Mixup, we can also infer that structural information is still highly retained in the mashed ego-graphs after Mixup is applied. The averaging operation in Mixup extracts an essential part of the structural information, which facilitates the training in the server and results in better performance of the model. Further, we observe that the performance marginally decreases without activation function in the personalization layers, which indicates that the model does not necessarily suffer from the bias introduced by the activation function, particularly in the case of extremely non-IID scenario.

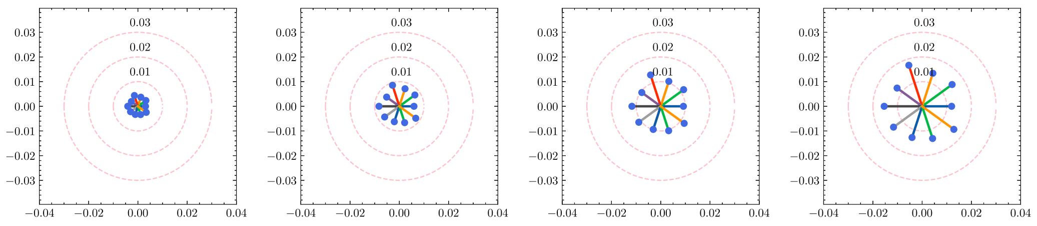
To further figure out whether it is worthwhile to build ego-graphs and verify the effectiveness of reduction layers and personalization layers, we compare FedEgo with three ablation methods: (1) *FedEgo w/o EGO*: In this method, we replace the reduction layers with GNN and the personalization layers with MLP. In this case, graph mining is performed locally and only the reduction embedding needs to be uploaded rather than the explicit structure of the mashed ego-graphs. (2) *FedEgo w/o RL*: In this method, clients will only update their personalization layers without performing the averaging updates in reduction layers. In this case, clients will encrypt the graph data in various ways, making it difficult to train the global model in the server. (3) *FedEgo w/o PL*: In this method, clients will only update their reduction layers without performing the mixing update in personalization layers. In this case, clients will not receive the help of the global model but only benefit from averaging update in reduction layers. Moreover, we also include FedAvg, since FedAvg is an averaging version of FedEgo w/o PL compared to FedEgo.

The poor performance of FedEgo w/o EGO, as can be seen from Figure 4, demonstrates the necessity to upload mashed ego-graphs for structural information flow in the federated framework. Without the mashed ego-graphs, it is rather difficult for the server to capture the structural information solely from the reduction embedding and further provide proper help for clients.

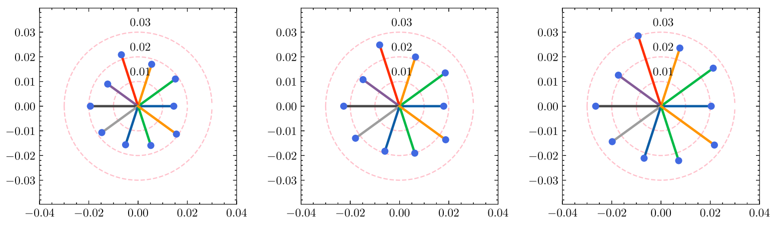


(a) Global test (b) Local test (c) Global test (d) Local test

Fig. 4. Ablation study on Cora (a,b) and Citeseer (c,d).



(a) $\gamma = 0.125$ (b) $\gamma = 0.25$ (c) $\gamma = 0.375$ (d) $\gamma = 0.5$



(e) $\gamma = 0.625$ (f) $\gamma = 0.75$ (g) $\gamma = 0.875$

Fig. 5. The visualization of weight divergence on Wiki.

Moreover, it is evidently clear that reduction layers play a more important part in enhancing the performance. Unsurprisingly, the removal of the averaging update in reduction layers hinders clients from encrypting the ego-graphs in the same way, and the mashed ego-graphs uploaded may be even harmful to the global model in early training. Besides, there is still a significant difference between FedEgo w/o PL and FedEgo, demonstrating the effectiveness of mixing updates in personalization layers. In the case of FedAvg, it requires the lowest convergence time due to its simple aggregation pattern. The slight gap between FedAvg and FedEgo w/o PL verifies the benefit brought by the averaging update in personalization layers, whereas the performance of FedAvg begins to decline after around 15 epochs before stabilizing in the global test. In the local test, however, the F1 score of FedAvg continues to increase until the end. As a result, FedAvg is inclined to improve clients' personalization ability at the expense of generalization ability. And finally, the observed difference between FedAvg and FedEgo in the local test is limited while there is a large gap in the global test, as we have discussed before.

## 5.7 Parameter Study

To have a better understanding of the hyperparameter $\gamma$, we estimate the weight divergence of each client with the varying $\gamma$. As shown in Figure 5, the global weight is fixed at the center of the circle, while scattered points represent the local weight of clients, and the distance to the center indicates the weight divergence. Circles with the same center but different radii are also plotted

Table 8. F1 Score on CoraFull with Different $\gamma$

| $\gamma$ | 0.125 | 0.25 | 0.375 | 0.5 | 0.625 | 0.75 | 0.875 |
|---|---|---|---|---|---|---|---|
| Local Test | 0.899 | 0.901 | 0.9 | 0.901 | 0.899 | **0.902** | 0.899 |
| Global Test | 0.652 | **0.653** | 0.648 | 0.649 | 0.649 | 0.648 | 0.646 |

Table 9. Communication Cost Analysis for 200 Epochs with Various Batch Sizes on Cora and Citeseer

| | Cora | | | | | | Citeseer | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | FedAvg | FedEgo | | | | | FedAvg | FedEgo | | | | |
| Batch Size | 32 | 8 | 16 | 32 | 64 | 128 | 32 | 8 | 16 | 32 | 64 | 128 |
| Global Test | 0.769 | 0.797 | 0.794 | 0.793 | 0.792 | 0.764 | 0.701 | 0.724 | 0.724 | 0.725 | 0.72 | 0.711 |
| Local Test | 0.952 | 0.958 | 0.959 | 0.958 | 0.955 | 0.945 | 0.917 | 0.92 | 0.919 | 0.918 | 0.918 | 0.917 |
| Time (mins) | 10.190 | 57.674 | 38.775 | 20.661 | 15.6 | 10.201 | 11.679 | 90.539 | 41.525 | 32.742 | 13.89 | 6.307 |
| Ego-graphs Cost (MB) | 0 | 6,071 | 3,055 | 1,545 | 790 | 425 | 0 | 6,895 | 3,477 | 1,769 | 899 | 471 |
| Parameters Cost (MB) | 6,819 | 2,801 | 2,801 | 2,801 | 2,801 | 2,801 | 11,250 | 7,234 | 7,234 | 7,234 | 7,234 | 7,234 |
| Total Cost (MB) | 6,819 | 8,872 | 5,856 | 4,346 | 3,591 | 3,226 | 11,250 | 14,129 | 10,711 | 9,003 | 8,133 | 7,705 |

for reference. As we can see from the result, the weight divergence is affected by $\gamma$ and becomes larger as $\gamma$ increases. According to Equation (15), $\lambda$ is negatively correlated with $\gamma$, and thus the weight divergence increases as $\lambda$ decreases, which is plausible support of Theorem 4.1. In addition to the visualization of weight divergence, we provide the F1 score over different values of $\gamma$ in Table 8. With a fine-tuned $\gamma$, the adaptive strategy is capable of discovering the best $\lambda$ for each client. Note that the optimal $\gamma$ for the generalization ability and the personalization ability are not always the same due to the statistical heterogeneity. With a relatively large deviation from the global ground-truth, reducing clients' generalization error will likely degrade their performance in the local dataset.

## 5.8 Analysis over Batch Size

We conduct extensive experiments and evaluate the communication cost in terms of **megabytes (MB)**. The results are provided under various batch sizes in Table 9.

The fact that smaller batches lead to higher performance and larger communication cost stands out in the results. The high quality of the uploaded mashed ego-graphs contributes to the improvement, while the communication cost and training over more data make the training time longer. The improvement of reducing batch size decreases as the batch size becomes smaller while the communication overhead, however, will be doubled due to more batches in the local training. Furthermore, a smaller batch size fails to meet the need for privacy protection. Therefore, we believe that selecting 32 as the batch size is a fair tradeoff based on the results. Additionally, the weight parameters mainly account for the communication cost with such a proper batch size, and it is acceptable to promote the performance at the expense of the additional cost.

## 5.9 Preserving Privacy with Gaussian Noise

As stated in Section 4.3, using a small batch size will result in privacy risks. To mitigate these issues, it is a common practice to add random Gaussian noise to the data to mask sensitive information while still preserving the overall statistical properties of the data, according to the knowledge of differential privacy [17]. Results obtained from both label-skew scenarios (Table 10) and the community clustering scenarios (Table 11) demonstrate the high robustness of FedEgo with respect to the Gaussian noise. Importantly, FedEgo still outperforms previous methods like FedSage+ when the standard deviation $\sigma$ is relatively large. In other words, we could combine FedEgo with the DP-related techniques to further preserve privacy without significantly compromising model performance.

Table 10. Performance of FedEgo with Gaussian Noise under Label-skew Scenarios

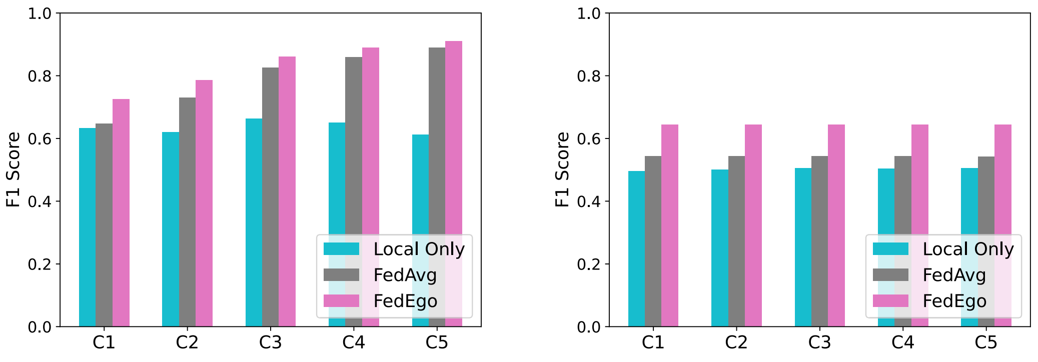
| | Global Test | | | | Local Test | | | |
|---|---|---|---|---|---|---|---|---|
| σ | Cora | Citeseer | Wiki | CoraFull | Cora | Citeseer | Wiki | CoraFull |
| 0 | 0.804 | 0.718 | 0.826 | 0.621 | 0.959 | 0.924 | 0.908 | 0.898 |
| 0.05 | 0.803 | 0.726 | 0.825 | 0.623 | 0.957 | 0.918 | 0.901 | 0.894 |
| 0.075 | 0.797 | 0.722 | 0.824 | 0.62 | 0.958 | 0.919 | 0.897 | 0.89 |
| 0.15 | 0.796 | 0.714 | 0.815 | 0.612 | 0.957 | 0.919 | 0.889 | 0.884 |
| 0.3 | 0.795 | 0.702 | 0.807 | 0.598 | 0.95 | 0.911 | 0.881 | 0.873 |

σ refers to the standard deviation of Gaussian noise.

Table 11. Performance of FedEgo with Gaussian Noise under Community Clustering Scenarios

| | Global Test | | | | | Local Test | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| σ | Cora | Citeseer | Wiki | CoraFull | FedDBLP | Cora | Citeseer | Wiki | CoraFull | FedDBLP |
| 0 | 0.754 | 0.689 | 0.813 | 0.49 | 0.829 | 0.724 | 0.713 | 0.81 | 0.512 | 0.785 |
| 0.05 | 0.747 | 0.692 | 0.808 | 0.488 | 0.828 | 0.726 | 0.714 | 0.806 | 0.501 | 0.782 |
| 0.075 | 0.749 | 0.692 | 0.801 | 0.487 | 0.829 | 0.724 | 0.714 | 0.799 | 0.502 | 0.78 |
| 0.15 | 0.741 | 0.689 | 0.799 | 0.472 | 0.833 | 0.716 | 0.709 | 0.793 | 0.486 | 0.767 |
| 0.3 | 0.744 | 0.684 | 0.792 | 0.466 | 0.831 | 0.724 | 0.708 | 0.785 | 0.438 | 0.76 |

σ refers to the standard deviation of Gaussian noise.



(a) F1 score in the local test

(b) F1 score in the global test

Fig. 6. F1 score of each client(C1–C5) on CoraFull.

## 5.10 Improvements for Each Client

To further explore how FedEgo improves the performance of a specific client, we provide the comparison results of each client in the local and the global test, as shown in Figure 6. The results indicate that FedEgo enhances the generalization ability and the personalization ability of all the clients. Compared to FedAvg, FedEgo offers more substantial benefits for clients. Moreover, the performance of clients in the global test is relatively stable, even though the graph data are severe non-IID among clients. The stable and significant improvement in performance demonstrates the effect of FedEgo.

## 5.11 Effort of Client Participation

In a real scenario, more clients in participation bring more information and data that may have a potential impact on the statistical heterogeneity. Therefore, we study the influence of client

Table 12. F1 Score Results in the Global Test on Wiki with Different Number of Clients

| Number of Clients | 5 | 10 | 15 | 20 | 25 | 30 | 35 |
|---|---|---|---|---|---|---|---|
| Local Test | **0.9261** | 0.9256 | 0.9245 | 0.9255 | 0.9254 | 0.9237 | 0.9247 |
| Global Test | 0.8179 | **0.8187** | 0.8175 | 0.8174 | 0.8169 | 0.8158 | 0.8157 |

Table 13. Analysis of Adaptive Mixing Coefficient

| | Global Test | | Local Test | |
|---|---|---|---|---|
| Methods | Cora | CoraFull | Cora | CoraFull |
| FedEgo w/o PL (Fixed $\lambda = 0$) | 0.7705 | 0.6173 | 0.9345 | 0.8488 |
| FedEgo (Fixed $\lambda = 0.5$) | 0.7843 | 0.6437 | 0.9585 | 0.9005 |
| FedEgo-Server (Fixed $\lambda = 1$) | 0.7921 | 0.6418 | 0.9535 | 0.8985 |
| FedEgo (Adaptive $\lambda$) | **0.7939** | **0.6442** | **0.959** | **0.9012** |

participation on FedEgo in this experiment. In particular, we select 5, 10, 15, 20, 30, and 35 clients to participate in each round on Wiki, and the experimental results are reported in Table 12. The personalization ability mainly depends on the client's own dataset and thus slightly drops with an increase in clients due to the introduced bias. As for generalization ability, the global model will have capability to collect more graph data to better fit the real dataset and further develop the generalization ability with more clients participating. This is especially the case when the number of clients is small. However, with a large number of clients, the benefit of introducing new clients diminishes, and the training time increases. Redundant clients might not boost the model performance when the number of clients is sufficient and the data is no longer in short supply.

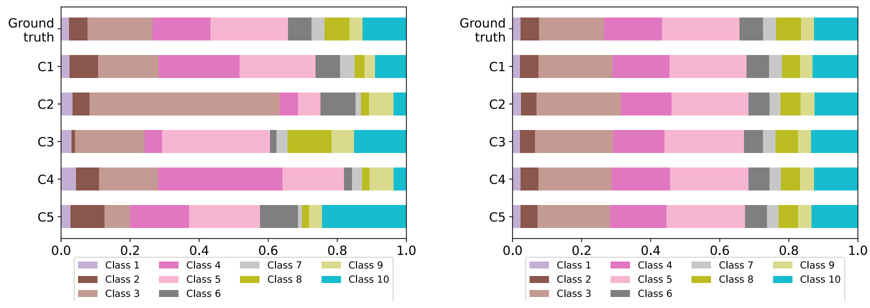
## 5.12 Benefit of Adaptive Mixing Coefficient

We further conduct additional experiments on Cora and CoraFull to verify whether FedEgo benefits from the adaptive mixing coefficient $\lambda$. We compare adaptive $\lambda$ with the following methods: (1) *FedEgo w/o PL*: As mentioned before, in this method, clients would not perform the mixing update in the personalization layers. Thus, it can be considered as a variant when $\lambda$ is fixed to be 0. (2) *FedEgo (Fixed $\lambda = 0.5$)*: In this method, we fixed $\lambda$ to be 0.5 when performing the mixing update in the personalization layers. (3) *FedEgo-Server*: In this method, clients entirely replace the personalization layers with the weight in the server for graph mining. Thus, it can be considered as a variant when $\lambda$ is fixed to be 1.

As can be seen from Table 13, adaptive $\lambda$ improves clients' performance to some extent. In the fixed pattern, FedEgo w/o PL has the worst performance without help of the global model. In the other extreme, FedEgo-Server is inferior to FedEgo with adaptive $\lambda$ due to the absence of the personalization. With regard to the compromise strategy, however, the fixed $\lambda = 0.5$ may be too small for some clients with larger *EMD* but too large for those with smaller *EMD*, preventing clients from achieving better performance. By contrast, the adaptive $\lambda$ allows clients to find their optimal level of participation in federated learning. With adaptive $\lambda$, clients perform better in the local test and achieve better personalization. Meanwhile, appropriate $\lambda$ for each client also improves their F1 score in the global test, which indicates that the mixing coefficient $\lambda$ in personalization layers has a significant influence on the generalization ability of the model.

## 5.13 Visualization

To comprehend how FedEgo improves the performance, we provide the original label distribution and the model prediction of FedEgo. The distribution of five local biased and the global testing



(a) Global ground-truth v.s. local ground-truth (b) Global ground-truth v.s. model predictions

Fig. 7. Label distribution of each client (C1–C5) on Wiki.

datasets is shown in Figure 7(a). Despite the fact that the graph data are severely non-IID, FedEgo enables clients to effectively extract useful information from their biased local datasets and establish collaboration. As a result, clients are capable of making relatively correct predictions following the global ground-truth, as shown in Figure 7(b).

## 6 CONCLUSION

In this article, we study the personalized federated node classification task on graphs and discuss three main challenges in a realistic setting. The proposed ego-graph-based federated framework FedEgo makes full use of structural information and tackles non-IID graph data by training a global model in the server. Moreover, clients adapt the global model to its local dataset by mixing the local and global weights. Besides, the Mixup technique is also applied for model robustness and privacy concern. Eventually, FedEgo outperforms baselines significantly with empirical evidence, showing its ability to address the difficult challenges in federated graph learning. Similar to existing FL methods, future work needs to address the communication cost for FedEgo, ideally focusing on efficient compression methods for ego-graphs and model weights.

## APPENDICES

## A PROOF OF THEOREM 4.1

In this section, we provide the detailed proof of Theorem 4.1.

Proof. Assume a batch of ego-graphs are in a fixed shape with $K$ hop neighbors and nodes in each layer extends $n$ neighbors. For convenience, we sort the nodes based on its layer to assign each node a specific position and obtain the position sets $Q_i$ in the $i$th layers, i.e., $Q_0 = \{0\}$, $Q_1 = \{1, 2, \dots n\}$, $Q_2 = \{n + 1, n + 2, \dots n^2\}$ and so on. The node in a specific position connects to its corresponding neighbors in the next layer. For example, the neighbors of node 1 in the second layer can be represented as $\{n+1, n+2, \dots 2n\}$. The position of nodes within the same layer can be assigned arbitrarily as long as the special connectivity between layers is maintained. Then, the sum of nodes' embedding in a specific layer does not change for different alignment when generating mashed ego-graph. Formally, given a batch of ego-graphs and the mashed ego-graph under an alignment $G$, let $p_i^{(0)}$ be the embedding of node $i$ in the mashed ego-graph, we have:

$$\sum_{i \in Q_j} p_i^{(0)} \equiv Sum_j, \forall G, \tag{16}$$

where $Sum_j$ equals the sum of original embedding in the $j$th layer of the sampled batch ego-graphs.

Then, a $K$ layer of GraphSAGE according to Equation (9) can be written into the following format:

$$P^{(K)} = A^K P^{(0)} W, \tag{17}$$

where $P$ is the embedding matrix, $A$ denotes the adjacency matrix of the mashed ego-graph, and $W$ is a reparameterized weight matrix by multiplication $W = W^{(1)} W^{(2)} \dots W^{(K)}$. In particular, $p_0^{(K)}$ denotes the final embedding of the center node.

For convenience, we further expand $P^{(0)}$ and $A^K$ as follows:

$$P^{(0)} = \begin{bmatrix} P_0^{(0)} \\ \vdots \\ P_{n^K}^{(0)} \end{bmatrix}, \tag{18}$$

$$A^K = \begin{bmatrix} A_{0,0}^K & \cdots & A_{0,n^K}^K \\ \vdots & \ddots & \vdots \\ A_{n^K,0}^K & \cdots & A_{n^K,n^K}^K \end{bmatrix}. \tag{19}$$

And thus, according to Equation (17), we have:

$$P^{(K)} = \begin{bmatrix} A_{0,0}^K & \cdots & A_{0,n^K}^K \\ \vdots & \ddots & \vdots \\ A_{n^K,0}^K & \cdots & A_{n^K,n^K}^K \end{bmatrix} \begin{bmatrix} P_0^{(0)} \\ \vdots \\ P_{n^K}^{(0)} \end{bmatrix} W. \tag{20}$$

We focus on the center node and compute its final embedding as follows:

$$P_0^{(K)} = \begin{bmatrix} A_{0,0}^K & \cdots & A_{0,n^K}^K \end{bmatrix} \begin{bmatrix} P_0^{(0)} \\ \vdots \\ P_{n^K}^{(0)} \end{bmatrix} W. \tag{21}$$

Due to symmetry of the mashed ego-graph, given a layer $j$, the weight of the edge between node 0 and each node $i$ in layer $j$ should be the same in the final adjacency matrix $A^K$. Thus, we have:

$$A_{0,i}^K \equiv \alpha_j, \forall i \in Q_j, \forall G, \tag{22}$$

where $\alpha_j$ is a constant that only depends on the $K$ and $n$.

We continue to combine Equations (16), (21), with (22) to calculate the final embedding of the center node and complete the proof:

$$p_0^{(K)} \equiv \left( \sum_j^K \alpha_j Sum_j \right) W, \forall G. \tag{23}$$

□

## B PROOF OF THEOREM 4.3

In this section, we provide the detailed proof of Theorem 4.3.

Proof. According to the definition of the $\Phi_i^{(T)}$ and $\Phi_g^{(T)}$, we further define the model of epoch $j$ after the $T$th update as $\Phi_{i,j}^{(T)}$ and $\Phi_{g,j}^{(T)}$. For simplicity, the update in personalization layers is defined as the $t$th step after the $T$th update. The weight after the update in personalization layers can be written as:

$$\begin{cases} \Phi_{i,t}^{(T)} = \lambda_i \Phi_{g,t-1}^{(T)} + (1-\lambda_i)\Phi_{i,t-1}^{(T)} \\ \Phi_{g,t}^{(T)} = \Phi_{g,t-1}^{(T)}. \end{cases} \tag{24}$$

And hence, we have the change of weight divergence after single update in personalization layers:

$$\begin{aligned} \|\Phi_{i,t}^{(T)} - \Phi_{g,t}^{(T)}\| =& \|\lambda_i \Phi_{g,t-1}^{(T)} + (1-\lambda_i)\Phi_{i,t-1}^{(T)} - \Phi_{g,t-1}^{(T)}\| \\ =& (1-\lambda_i)\|\Phi_{g,t-1}^{(T)} - \Phi_{g,t-1}^{(T)}\|. \end{aligned} \tag{25}$$

Now, take the SGD update of the clients and the server into consideration. Given cross-entropy loss defined as Equation (2), SGD update in the $t-1$-th step performs:

$$\begin{cases} \Phi_{i,t-1}^{(T)} = \Phi_{i,t-2}^{(T)} - \eta \nabla_\Phi \ell_i \left(\Phi_{i,t-2}^{(T)}\right) \\ \Phi_{g,t-1}^{(T)} = \Phi_{g,t-2}^{(T)} - \eta \nabla_\Phi \ell_g \left(\Phi_{g,t-2}^{(T)}\right). \end{cases} \tag{26}$$

Therefore, we have:

$$\begin{cases} \Phi_{i,t-1}^{(T)} = \Phi_{i,t-2}^{(T)} - \eta \sum_{c=1}^{C} P_i(y=c) \nabla_\Phi \mathbb{E}_{x|y=c}[\log f_c(x, \Phi_{i,t-2}^{(T)})] \\ \Phi_{g,t-1}^{(T)} = \Phi_{g,t-2}^{(T)} - \eta \sum_{c=1}^{C} P_g(y=c) \nabla_\Phi \mathbb{E}_{x|y=c}[\log f_c(x, \Phi_{g,t-2}^{(T)})]. \end{cases} \tag{27}$$

We next calculate the weight divergence as follows:

$$\begin{aligned} &\|\Phi_{i,t-1}^{(T)} - \Phi_{g,t-1}^{(T)}\| \\ =& \Bigg\| \Phi_{i,t-2}^{(T)} - \eta \sum_{c=1}^{C} P_i(y=c) \nabla_\Phi \mathbb{E}_{x|y=c}\left[\log f_c\left(x, \Phi_{g,t-2}^{(T)}\right)\right] \\ &\quad - \Phi_{g,t-2}^{(T)} + \eta \sum_{c=1}^{C} P_g(y=c) \nabla_\Phi \mathbb{E}_{x|y=c}\left[\log f_c\left(x, \Phi_{g,t-2}^{(T)}\right)\right] \Bigg\| \\ \le& \left\|\Phi_{i,t-2}^{(T)} - \Phi_{g,t-2}^{(T)}\right\| \\ &\quad + \eta \Bigg\| \sum_{c=1}^{C} P_i(y=c) \Bigg( \nabla_w \mathbb{E}_{x|y=c}\left[\log f_c\left(x, \Phi_{i,t-2}^{(T)}\right)\right] \\ &\qquad - \nabla_w \mathbb{E}_{x|y=c}\left[\log f_c\left(x, \Phi_{g,t-2}^{(T)}\right)\right] \Bigg) \\ &\quad + \eta \sum_{c=1}^{C} \left(P_i(y=c) - P_g(y=c)\right) \nabla_w \mathbb{E}_{x|y=c}\left[\log f_c\left(x, \Phi_{g,t-2}^{(T)}\right)\right] \Bigg\| \\ \le& \left\|\Phi_{i,t-2}^{(T)} - \Phi_{g,t-2}^{(T)}\right\| \\ &\quad + \eta \Bigg\| \sum_{c=1}^{C} P_i(y=c) \Bigg( \nabla_w \mathbb{E}_{x|y=c}\left[\log f_c\left(x, \Phi_{i,t-2}^{(T)}\right)\right] \\ &\qquad - \nabla_w \mathbb{E}_{x|y=c}\left[\log f_c\left(x, \Phi_{g,t-2}^{(T)}\right)\right] \Bigg) \Bigg\| \\ &\quad + \eta \left\| \sum_{c=1}^{C} \left(P_i(y=c) - P_g(y=c)\right) \nabla_w \mathbb{E}_{x|y=c}\left[\log f_c\left(x, \Phi_{g,t-2}^{(T)}\right)\right] \right\|. \end{aligned} \tag{28}$$

The last inequality holds due to absolute value inequality.

For simplicity, we denote the gradient of $\mathbb{E}_{x|y=c}\left[\log f_c\left(x, \Phi_{i,t-2}^{(T)}\right)\right]$ of label $c$ as $g_c$, which only depends on model weight $\Phi$:

$$g_c(\Phi) = \nabla_w \mathbb{E}_{x|y=c}\left[\log f_c\left(x, \Phi\right)\right]. \tag{29}$$

Since $g_c(\Phi) = \nabla_w \mathbb{E}_{x|y=c} \log f_c(x, \Phi)$ is Ł$_{x|y=c}$-Lipschitz, we have:

$$\left\|g_c(\Phi_i^{(T)}) - g_c(\Phi_g^{(T)})\right\| \leq L_{x|y=c}\left\|\Phi_i^{(T)} - \Phi_g^{(T)}\right\|. \tag{30}$$

Then, we continue to compute the weight divergence based on Equations (29), (30):

$$\begin{aligned}
&\left\|\Phi_{i,t-1}^{(T)} - \Phi_{g,t-1}^{(T)}\right\| \\
\leq& \left(1 + \eta \sum_{c=1}^{C} P_i(y=c) L_{x|y=c}\right)\left\|\Phi_{i,t-2}^{(T)} - \Phi_{g,t-2}^{(T)}\right\| \\
&\quad + \eta \left\|\sum_{c=1}^{C}\left(P_i(y=c) - P_g(y=c)\right) g_c(\Phi_{g,t-2}^{(T)})\right\| \\
\leq& \left(1 + \eta L_{max} \sum_{c=1}^{C} P_i(y=c)\right)\left\|\Phi_{i,t-2}^{(T)} - \Phi_{g,t-2}^{(T)}\right\| \\
&\quad + \eta g_{max}(\Phi_{g,t-2}^{(T)}) \sum_{c=1}^{C}\left\|P_i(y=c) - P_g(y=c)\right\| \\
=& \left(1 + \eta L_{max}\right)\left\|\Phi_{i,t-2}^{(T)} - \Phi_{g,t-2}^{(T)}\right\| + \eta g_{max}(\Phi_{g,t-2}^{(T)}) \sum_{c=1}^{C}\left\|P_i(y=c) - P_g(y=c)\right\|,
\end{aligned} \tag{31}$$

where $L_{max} = \max_{c=1}^{C} L_{x|y=c}$ and $g_{\max}$ is defined as follows:

$$g_{\max}\left(\Phi_{g,t-2}^{(T)}\right) = \max_{c=1}^{C} g_c\left(\Phi_{g,t-2}^{(T)}\right) = \max_{c=1}^{C}\left\|\nabla_w \mathbb{E}_{x|y=c} \log f_c\left(x, \Phi_{g,t-2}^{(T)}\right)\right\|. \tag{32}$$

We further define $a = 1 + \eta L_{max}$ and we have the following inequality for t-1 steps SGD update by induction:

$$\begin{aligned}
&\left\|\Phi_{i,t-1}^{(T)} - \Phi_{g,t-1}^{(T)}\right\| \\
\leq& a\left\|\Phi_{i,t-2}^{(T)} - \Phi_{g,t-2}^{(T)}\right\| + \eta g_{\max}\left(\Phi_{g,t-2}^{(T)}\right) \sum_{c=1}^{C}\left\|P_i(y=c) - P_g(y=i)\right\| \\
\leq& a^2\left\|\Phi_{g,t-3}^{(T)} - \Phi_{i,t-3}^{(T)}\right\| \\
&\quad + \eta \sum_{c=1}^{C}\left\|P_i(y=c) - P_g(y=i)\right\|\left(g_{\max}\left(\Phi_{g,t-2}^{(T)}\right) + a g_{\max}\left(\Phi_{g,t-3}^{(T)}\right)\right) \\
\leq& a^{t-1}\left\|\Phi_{i,0}^{(T)} - \Phi_{g,0}^{(T)}\right\| \\
&\quad + \eta \sum_{c=1}^{C}\left\|P_i(y=c) - P_g(y=i)\right\|\left(\sum_{j=0}^{t-2} a^j g_{\max}\left(\Phi_{g,t-2-j}^{(T)}\right)\right) \\
=& a^{t-1}\left\|\Phi_{g,t}^{(T-1)} - \Phi_{g,t}^{(T-1)}\right\| \\
&\quad + \eta \sum_{c=1}^{C}\left\|P_i(y=c) - P_g(y=i)\right\|\left(\sum_{j=0}^{t-2} a^j g_{\max}\left(\Phi_{g,t-2-j}^{(T)}\right)\right).
\end{aligned} \tag{33}$$

Combining Equation (25) with Equation (33), we bound the weight divergence after $T$th update and complete the proof:

$$\begin{aligned}
&\left\|\Phi_{i,t}^{(T)} - \Phi_{g,t}^{(T)}\right\| \\
=&(1-\lambda_i)\left\|\Phi_{g,t-1}^{(T)} - \Phi_{g,t-1}^{(T)}\right\| \\
\leq&(1-\lambda_i)a^{t-1}\left\|\Phi_{g,t}^{(T-1)} - \Phi_{g,t}^{(T-1)}\right\| \\
&+\eta(1-\lambda_i)\sum_{c=1}^{C}\left\|P_i(y=c) - P_g(y=i)\right\|\left(\sum_{j=0}^{t-2} a^j g_{\max}\left(\Phi_{g,t-2-j}^{(T)}\right)\right).
\end{aligned} \tag{34}$$

□

## REFERENCES

[1] Manoj Ghuhan Arivazhagan, Vinay Aggarwal, Aaditya Kumar Singh, and Sunav Choudhary. 2019. Federated learning with personalization layers. *arXiv preprint arXiv:1912.00818* (2019).

[2] Lu Bai and Edwin R. Hancock. 2016. Fast depth-based subgraph kernels for unattributed graphs. *Pattern Recog.* 50 (2016), 233–245.

[3] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *J. Stat. Mechan.: Theor. Experim.* 2008, 10 (2008), P10008.

[4] Aleksandar Bojchevski and Stephan Günnemann. 2017. Deep Gaussian embedding of graphs: Unsupervised inductive learning via ranking. *arXiv preprint arXiv:1707.03815* (2017).

[5] Chen Cai and Yusu Wang. 2020. A note on over-smoothing for graph neural networks. *arXiv preprint arXiv:2006.13318* (2020).

[6] Chuan Chen, Weibo Hu, Ziyue Xu, and Zibin Zheng. 2021. FedGL: Federated graph learning framework with global self-supervision. *arXiv preprint arXiv:2105.03170* (2021).

[7] Fei Chen, Mi Luo, Zhenhua Dong, Zhenguo Li, and Xiuqiang He. 2018. Federated meta-learning with fast convergence and efficient communication. *arXiv preprint arXiv:1802.07876* (2018).

[8] Liam Collins, Hamed Hassani, Aryan Mokhtari, and Sanjay Shakkottai. 2021. Exploiting shared representations for personalized federated learning. *arXiv preprint arXiv:2102.07078* (2021).

[9] Yuyang Deng, Mohammad Mahdi Kamani, and Mehrdad Mahdavi. 2020. Adaptive personalized federated learning. *arXiv preprint arXiv:2003.13461* (2020).

[10] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Conference on Advances in Neural Information Processing Systems*. 1024–1034.

[11] Filip Hanzely and Peter Richtárik. 2020. Federated learning of a mixture of global and local models. *arXiv preprint arXiv:2002.05516* (2020).

[12] Chaoyang He, Keshav Balasubramanian, Emir Ceyani, Carl Yang, Han Xie, Lichao Sun, Lifang He, Liangwei Yang, Philip S. Yu, Yu Rong, Peilin Zhao, Junzhou Huang, Murali Annavaram, and Salman Avestimehr. 2021. FedGraphNN: A federated learning system and benchmark for graph neural networks. *arXiv preprint arXiv:2104.07145* (2021).

[13] Yihan Jiang, Jakub Konečnỳ, Keith Rush, and Sreeram Kannan. 2019. Improving federated learning personalization via model agnostic meta learning. *arXiv preprint arXiv:1909.12488* (2019).

[14] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. Federated optimization in heterogeneous networks. *Proc. Mach. Learn. Syst.* 2 (2020), 429–450.

[15] Yishay Mansour, Mehryar Mohri, Jae Ro, and Ananda Theertha Suresh. 2020. Three approaches for personalization with applications to federated learning. *arXiv preprint arXiv:2002.10619* (2020).

[16] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*. PMLR, 1273–1282.

[17] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. 2017. Learning differentially private recurrent language models. *arXiv preprint arXiv:1710.06963* (2017).

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

[19] Yang Pei, Renxin Mao, Yang Liu, Chaoran Chen, Shifeng Xu, Feng Qiang, and Blue Elephant Tech. 2021. Decentralized federated graph neural networks. In *International Workshop on Federated and Transfer Learning for Data Sparsity and Confidentiality in Conjunction with IJCAI*.

[20] Yeqing Qiu, Chenyu Huang, Jianzong Wang, Zhangcheng Huang, and Jing Xiao. 2022. A privacy-preserving subgraph-level federated graph neural network via differential privacy. In *15th International Conference on Knowledge Science, Engineering and Management (KSEM'22)*. Springer, 165–177.

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

[22] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet Talwalkar. 2017. Federated multi-task learning. *arXiv preprint arXiv:1705.10467* (2017).

[23] Binghui Wang, Ang Li, Hai Li, and Yiran Chen. 2020. GraphFL: A federated learning framework for semi-supervised node classification on graphs. *arXiv preprint arXiv:2012.04187* (2020).

[24] Zhen Wang, Weirui Kuang, Yuexiang Xie, Liuyi Yao, Yaliang Li, Bolin Ding, and Jingren Zhou. 2022. FederatedScope-GNN: Towards a unified, comprehensive and efficient package for federated graph learning. In *28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 4110–4120.

[25] Han Xie, Jing Ma, Li Xiong, and Carl Yang. 2021. Federated graph classification over non-IID graphs. *Adv. Neural Inf. Process. Syst.* 34 (2021).

[26] Yuhang Yao and Carlee Joe-Wong. 2022. FedGCN: Convergence and communication tradeoffs in federated training of graph convolutional networks. *arXiv preprint arXiv:2201.12433* (2022).

[27] Tehrim Yoon, Sumin Shin, Sung Ju Hwang, and Eunho Yang. 2021. FedMix: Approximation of mixup under mean augmented federated learning. *arXiv preprint arXiv:2107.00233* (2021).

[28] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. 2017. Mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412* (2017).

[29] Ke Zhang, Carl Yang, Xiaoxiao Li, Lichao Sun, and Siu Ming Yiu. 2021. Subgraph federated learning with missing neighbor generation. *Adv. Neural Inf. Process. Syst.* 34 (2021).

[30] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. 2018. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582* (2018).

[31] Qi Zhu, Yidan Xu, Haonan Wang, Chao Zhang, Jiawei Han, and Carl Yang. 2020. Transfer learning of graph neural networks with ego-graph information maximization. *arXiv preprint arXiv:2009.05204* (2020).

Received 8 September 2022; revised 25 June 2023; accepted 30 August 2023