Subgraph Federated Learning for Local Generalization

Sungwon Kim
KAIST
swkim@kaist.ac.kr

Yoonho Lee
KAIST
sml0399benbm@kaist.ac.kr

Yunhak Oh
KAIST
yunhak.oh@kaist.ac.kr

Namkyeong Lee
KAIST
namkyeong96@kaist.ac.kr

Sukwon Yun
UNC Chapel Hill
swyun@cs.unc.edu

Junseok Lee
KAIST
junseoklee@kaist.ac.kr

Sein Kim
KAIST
rlatpdlsgns@kaist.ac.kr

Carl Yang
Emory University
j.carlyang@emory.edu

Chanyoung Park*
KAIST
cy.park@kaist.ac.kr

ABSTRACT

Federated Learning (FL) on graphs enables collaborative model training to enhance performance without compromising the privacy of each client. However, previous methods often overlook the mutable nature of graph data, which frequently introduces new nodes and leads to shifts in label distribution. Unlike prior methods that struggle to generalize to unseen nodes with diverse label distributions, our proposed method, FedLoG, effectively addresses this issue by alleviating the problem of local overfitting. Our model generates global synthetic data by condensing the reliable information from each class representation and its structural information across clients. Using these synthetic data as a training set, we alleviate the local overfitting problem by adaptively generalizing the absent knowledge within each local dataset. This enhances the generalization capabilities of local models, enabling them to handle unseen data effectively. Our model outperforms the baselines in proposed experimental settings, which are designed to measure generalization power to unseen data in practical scenarios. Our code is available at https://github.com/sung-won-kim/FedLoG

KEYWORDS

Subgraph Federated Learning; Graph Neural Network; Local Overfitting

1 INTRODUCTION

In the realm of Graph Neural Networks (GNNs), most systems are designed for a unified, centralized graph. However, real-world applications [21] frequently involve individual users or institutions maintaining private graphs, isolated due to privacy concerns. Graph Federated Learning (GFL) [10] provides a solution by enabling clients to independently train local GNNs on their data. This decentralized training approach allows a central server to aggregate the locally updated weights from multiple clients, creating a unified model that respects privacy constraints. In this paper, we explore a particularly challenging aspect of GFL—distributed subgraphs (subgraph-FL), where clients manage largely disjoint sets of nodes and their links.

In real-world scenarios, graph data frequently changes, particularly in social, citation, and e-commerce networks [12, 14, 15]. These changes often result in new label distribution patterns that are distinct from the existing local label distribution. Despite this, existing subgraph-FL methods [2, 17, 18, 21] primarily focus on optimizing models based on the current label distribution within each client (i.e., local optimization). However, some studies [4, 9, 20] demonstrate that client models are particularly prone to local overfitting after local updates, resulting in a significant decrease in the accuracy of minority classes (i.e., tail classes) within the local data. Given these limitations, current approaches encounter significant practical challenges, particularly in adapting to new nodes added to the original local graph, especially those in tail or unseen classes which are not present in the local graph but existing in others (i.e., missing classes). These nodes, which form new connections with existing nodes, often have structural patterns unfamiliar to local clients, leading to substantial discrepancies in both label and structural distributions.

Existing methods in FL [4, 9, 20] aim to ensure that local models can make predictions for all classes without bias by mitigating local overfitting caused by the local label distribution. Specifically, they propose regularizing the logits of each class in the local models to align more closely with those of the global model. While these methods effectively address local overfitting and manage tail or missing classes, increasing the logits of local tail data risks amplifying noisy data, which is harmful for the class representation of the global model. Beyond FL, another approach to mitigating the problem of overfitting on train data involves addressing class imbalance by reducing the long-tail class distribution. Techniques such as down-sampling [6], over-sampling [22], or constructing expert models for long-tailed data [19] are commonly used. Despite their effectiveness, they require at least one data point to be present for each class, facing challenges when a class is missing in a local client while present in others.

In this paper, we propose to address the local overfitting issue of subgraph-FL by introducing reliable global synthetic data that **1)** learn accurate class representations, and **2)** mitigate class imbalance, including missing classes. Specifically, we aggregate knowledge from local data across all clients for each class and integrate it into

*Corresponding author.

FedKDD '24, August 26, 2024, Barcelona, Spain.

the global synthetic data. Subsequently, each client adaptively utilizes the global synthetic data as additional training data to ensure effective learning for all classes, including those that are underrepresented or missing in each client. This strategy helps prevent local overfitting even after local updates and enables accurate class representation (i.e., local generalization). However, there exist two crucial challenges that need consideration:

C1. Which data across all clients should be aggregated to ensure reliability? Since clients heavily rely on the knowledge from other clients to learn locally absent knowledge in FL, it becomes crucial to obtain and share knowledge from reliable data within each local graph. Here, data reliability refers to the accuracy and consistency of information sourced from decentralized nodes.

C2. How can data from other clients be utilized without compromising privacy? While direct sharing of the data between clients prevents local overfitting, it raises severe privacy concerns. Furthermore, directly using training data from all clients incurs high communication costs.

For the above two challenges, we propose the following solutions:

Solution to C1. Knowledge from Head degree and Head class nodes. We find that nodes with a high number of connected edges (i.e., head degree) and that belong to the majority class (i.e., head class) possess reliable structural and class representative information, significantly influencing the model generalization ability to unseen data. Motivated by these insights, we gather knowledge from clients and filter it based on their headness in terms of both degree and class.

Solution to C2. Data condensation. We propose to condense only reliable knowledge into synthetic data to share across the clients, thereby avoiding the direct use of individual client data while also minimizing the amount of data transferred between the server and local clients.

In summary, we propose a subgraph **Fed**erated Learning framework for **Lo**cal **G**eneralization, FedLoG, that generates global synthetic data with a novel reliable knowledge condensation strategy. This approach reduces the risk of noise in class representations, and enables each client to compensate for locally absent knowledge without compromising privacy. By doing so, FedLoG prevents local overfitting and ensures a well-generalized representation of all classes, enabling the successful handling of unseen data, including missing classes.

In this paper, we make the following contributions:

- We introduce FedLoG, the first work in subgraph-FL that focuses on preventing local overfitting, including the issue of missing classes, to address the mutable nature of the graph domain. This approach enhances the performance of the global model and the generalization power of local models, enabling them to effectively address unseen data for all classes.
- We analyze what constitutes reliable data in graph-based federated learning and propose a method to condense and share this knowledge across clients. This approach not only leverages reliable data effectively but also protects privacy by using condensed synthetic data.
- We propose practical evaluation settings for subgraph-FL, enabling measurement of the model's generalization on future data, assessing robustness in mutable graph domains, and demonstrating consistent outperformance over other baselines.

2 PRELIMINARIES

Related works are provided in Appendix A.

Notations. We use $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ to denote a graph with the set of nodes $\mathcal{V}$ and the set of edges $\mathcal{E}$. The dataset $\mathcal{D} = (\mathcal{G}, Y)$ includes labels Y for the nodes, and $X_{\mathcal{V}} \in \mathbb{R}^{|\mathcal{V}|\times d}$ is the feature matrix with d as the feature dimension. Each node $v \in \mathcal{V}$ has a feature vector $x_v \in \mathbb{R}^d$. The nodes are classified into $|C_{\mathcal{V}}|$ distinct classes. In subgraph-FL, a server S and K clients manage disjoint subgraphs $\mathcal{G}_k = (\mathcal{V}_k, \mathcal{E}_k)$ for each client k. The global set of nodes is $\mathcal{V} = \bigcup_{k=1}^{K} \mathcal{V}_k$ with $\mathcal{V}_i \cap \mathcal{V}_j = \varnothing$ for all $i \neq j$. The local dataset for client k is $\mathcal{D}_k = (\mathcal{G}_k, Y_k)$, and the combined local datasets are $\mathcal{D}_{\text{local}} = \bigcup_{k=1}^{K} \mathcal{D}_k$. Additionally, we generate a global synthetic set $\mathcal{D}_g = (\mathcal{G}_g, Y_g)$, where $\mathcal{G}_g = (\mathcal{V}_g, \mathcal{E}_{\varnothing})$ consists of isolated nodes $v_g \in \mathcal{V}_g$ with no edges $\mathcal{E}_{\varnothing}$. $\mathcal{V}_g$ includes s nodes per class, totaling $s\times |C_{\mathcal{V}}|$ nodes. A summary of the notations is provided in Appendix N.

Problem Statement. We aim to develop a distributed learning framework for collaborative training of a node classifier. Specifically, the classifier F uses optimized parameters ϕ to minimize a predefined task loss. The objective is to find global parameters ϕ^* that minimizes the aggregated local empirical risk $\mathcal{R}$, defined as: $\phi^* = \arg\min_{\phi} \mathcal{R}(F(\phi)) = \frac{1}{K}\sum_{k=1}^{K} \mathcal{R}_k(F_k(\phi))$, where $\mathcal{R}_k(F_k(\phi)) := \mathbb{E}_{(\mathcal{G}_k, Y_k)\sim\mathcal{D}_{\text{local}}}[\mathcal{L}_k(F_k(\phi;\mathcal{G}_k), Y_k)]$ and the task-specific loss $\mathcal{L}_k$ is defined as

$$\mathcal{L}_k := \frac{1}{|\mathcal{V}_k|}\sum_{v_k\in\mathcal{V}_k} l(\phi;\mathcal{G}_k(v_k), y_{v_k}) + \frac{1}{|\mathcal{V}_g|}\sum_{v_g\in\mathcal{V}_g} l(\phi; v_g, y_{v_g}).$$

To allow each client to generalize across all classes, including missing classes, we generate global synthetic data $\mathcal{D}_g$ and introduce an additional loss term (i.e., the second term) to take into account this data to prevent local overfitting.

Data Reliability. Data reliability refers to the accuracy and consistency of information from decentralized nodes, which is essential for training models across varied environments (i.e., clients). Our analysis shows that both head-degree and head-class nodes are reliable. Detailed analysis of the reliability of these nodes is provided in Appendix C.

3 METHODOLOGY

Our proposed subgraph-FL framework, FedLoG, condenses data from both head-degree and head-class nodes within each local graph to gather reliable knowledge from distributed graph data. This condensed data is then used as an additional training dataset for all clients, along with their local graph data, to prevent local overfitting. Our method operates as follows:

- **Step 1 – Local Fitting (Section 3.1):** The server initializes the local model parameters of K clients with the parameters of the global model ϕ. Each local model is then trained using local data $\mathcal{D}_k$. Concurrently, head degree and tail degree knowledge are condensed into synthetic nodes within each client, denoted as $\mathcal{V}_{k,\text{head}}$ and $\mathcal{V}_{k,\text{tail}}$.
- **Step 2 – Global Aggregation and Global Synthetic Data Generation (Section 3.2):** After local training, the server aggregates the local models to create the global model ϕ, and generates global synthetic data $\mathcal{D}_g$ by aggregating $\mathcal{V}_{k,\text{head}}$ for all k, weighted by the proportion of the head classes within each client k.
- **Step 3 – Local Fitting (Section 3.1) & Local Generalization (Section 3.3):** Similar to Step 1, local fitting and data condensation proceed. After local fitting, local models are generalized using

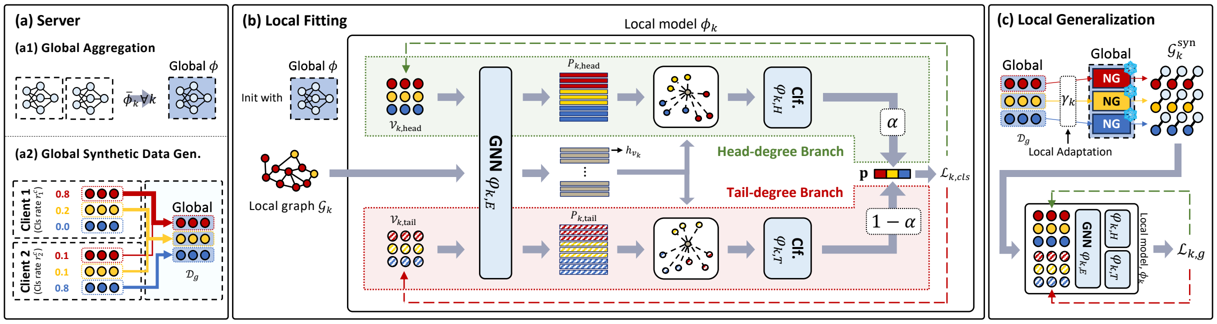

Figure 1: Overview of FedLoG with 2 Clients and 3 Classes.

$\mathcal{D}_g$ which possess both head degree and head class knowledge, adaptively learning the locally absent knowledge.

While the framework starts with Step 1, it continues to alternate between Steps 2 and 3 until the final round R is reached. In summary, our method extracts **head degree** knowledge at the **client** level and **head class** knowledge at the **server** level, then condense them into the global synthetic data, which is utilized to train the local model during Local Generalization to adaptively compensate for the locally absent knowledge within each client. The overall framework of FedLoG is depicted in Figure 1.

3.1 Local Fitting

The local model for each client k consists of one GNN embedder ($\varphi_{k,E}$) and two classifiers ($\varphi_{k,H}$ and $\varphi_{k,T}$), for the head and tail degree branches, respectively, as shown in Figure 1(b). Each branch has $s \times |C_{\mathcal{V}}|$ learnable nodes, each with features of dimension d, allocating s nodes per class. Thus, each client has learnable node sets $\mathcal{V}_{k,\text{head}}$ and $\mathcal{V}_{k,\text{tail}}$ with features $X_{\mathcal{V}_{k,\text{head}}} \in \mathbb{R}^{(s\times|C_{\mathcal{V}}|)\times d}$ and $X_{\mathcal{V}_{k,\text{tail}}} \in \mathbb{R}^{(s\times|C_{\mathcal{V}}|)\times d}$, respectively.

At the client level, we condense knowledge from locally observed nodes into these learnable nodes. Head degree nodes are condensed into $\mathcal{V}_{k,\text{head}}$ and tail degree nodes into $\mathcal{V}_{k,\text{tail}}$, integrating condensation and prediction into a single process. Learnable nodes serve as prototypes within each branch, forming a prototypical network. The detailed processes are as follows:

(Initialization) In the initial round ($r = 0$), we initialize the local model weights for each client k, denoted as $\phi_k = \{\varphi_{k,E}, \varphi_{k,H}, \varphi_{k,T}\}$, with the global set of parameters $\phi = \{\varphi_E, \varphi_H, \varphi_T\}$.

(Embedding) For each node $v_k \in \mathcal{V}_k$ with initial features $h^{(0)}_{v_k} = x_{v_k}$, a shared GraphSAGE [5] GNN encoder $\varphi_{k,E}$ is employed to embed the local node $v_k \in \mathcal{V}_k$ and the learnable nodes $v_{k,\text{head}} \in \mathcal{V}_{k,\text{head}}$ and $v_{k,\text{tail}} \in \mathcal{V}_{k,\text{tail}}$:

$$h_{v_k} = \mathbf{GNN}_{\varphi_{k,E}}(v_k, \mathcal{G}_k), \quad h_{v_{k,\text{head}}} = \mathbf{GNN}_{\varphi_{k,E}}(v_{k,\text{head}}, \mathcal{G}_I), \quad h_{v_{k,\text{tail}}} = \mathbf{GNN}_{\varphi_{k,E}}(v_{k,\text{tail}}, \mathcal{G}_I) \tag{1}$$

where h_{v_k}, $h_{v_{k,\text{head}}}$, and $h_{v_{k,\text{tail}}}$ are the representations of nodes v_k, $v_{k,\text{head}}$, and $v_{k,\text{tail}}$, respectively, and $\mathcal{G}_I$ represents a discrete graph with no edges. The learnable nodes do not adhere to a specific graph structure but share the same GNN encoder with the local graph, allowing us to condense structural information into the features of the learnable nodes.

After acquiring node representations, we generate model predictions in each branch using class prototypes, which are the representations of learnable nodes. For instance, in the head branch, prototypes are defined as

$$P_{k,\text{head}} = \{h_{v^{(1,1)}_{k,\text{head}}}, \ldots, h_{v^{(1,s)}_{k,\text{head}}}, \ldots, h_{v^{(|C_{\mathcal{V}}|,1)}_{k,\text{head}}}, \ldots, h_{v^{(|C_{\mathcal{V}}|,s)}_{k,\text{head}}}\}, \tag{2}$$

with s prototypes per class. To ensure all class information contributes to the final prediction, the target node representations are further updated based on feature differences with all prototypes assigned to each class as follows:

$$h'_{v_k} = \varphi_{k,H}(h_{v_k}, \{h_{v_k} - h_{v^{(1,1)}_{k,\text{head}}}, \ldots, h_{v_k} - h_{v^{(|C_{\mathcal{V}}|,s)}_{k,\text{head}}}\}). \tag{3}$$

Please refer to Appendix D for more details on $\varphi_{k,H}$. Then, the class probability for target node v_k is given as follows:

$$p(c|h'_{v_k}) = \frac{\exp(-d(h'_{v_k}, \bar{h}_{\mathcal{V}^c_{k,\text{head}}}))}{\sum_{c'=1}^{|C_{\mathcal{V}}|} \exp(-d(h'_{v_i}, \bar{h}_{\mathcal{V}^{c'}_{(k,\text{head})}}))}, \tag{4}$$

where $d(\cdot,\cdot)$ is the squared Euclidean distance and $\bar{h}_{\mathcal{V}^c_{k,\text{head}}}$ indicates the average of prototypes of class c, i.e., $\bar{h}_{\mathcal{V}^c_{k,\text{head}}} = \frac{1}{s}\sum_{i=1}^{s} h_{v^{(c,i)}_{k,\text{head}}}$.

To obtain final prediction $\mathbf{p}$, we combine the class probabilities from both branches $\mathbf{p}_{\text{head}}$ and $\mathbf{p}_{\text{tail}}$ by weighting them based on the degree value of the target node v_k (i.e., $\deg(v_k)$) as follows:

$$\mathbf{p} = \alpha \cdot \mathbf{p}_{\text{head}} + (1-\alpha) \cdot \mathbf{p}_{\text{tail}}, \tag{5}$$

where $\alpha = 1/(1 + e^{-(\deg(v_k)-(\lambda+1))})$, and λ is the tail degree threshold outlined in the Appendix E. Note that, the parameter α balances the influence of head and tail branches based on the node's degree, preventing nodes with very high degree from dominating the head branch. High-degree nodes, which rely heavily on the head degree branch for predictions, significantly influence the features of learnable nodes within the head degree branch, condensing their knowledge into $\mathcal{V}_{k,\text{head}}$. The tail degree branch specializes in classifying tail degree nodes, which possess knowledge that is not suitable for sharing across clients due to their potentially noisy characteristics [16]. Then, the prediction loss for each client k is calculated as follows: $\mathcal{L}_{k,cls} = \sum_{v_k\in\mathcal{V}_k}\sum_{c\in C_{\mathcal{V}}} -\mathbb{I}(y_{v_k} = c)\log(\mathbf{p}[c])$.

Furthermore, to ensure the stability of the condensation process, we minimize the L_2 norm of the learnable features, denoted as $\mathcal{L}_{k,norm}$. Thus, the total loss for model parameters is

$$\mathcal{L}_k(\phi_k, X_{\mathcal{V}_{k,\text{head}}}, X_{\mathcal{V}_{k,\text{tail}}}) = \mathcal{L}_{k,cls} + \beta \cdot \mathcal{L}_{k,norm}, \tag{6}$$

where β adjusts the extent of regularization.

3.2 Global Aggregation and Global Synthetic Data Generation

Global Aggregation. In Figure 1(a), after training the K local clients, the server aggregates the local model weights for round

Table 1: Model performance across settings. Mean accuracy with standard deviation over 3 runs.

(a) Seen Graph

	Cora			CiteSeer			PubMed			Amazon Photo			Amazon Computers		
Methods	3 Clients	5 Clients	10 Clients	3 Clients	5 Clients	10 Clients	3 Clients	5 Clients	10 Clients	3 Clients	5 Clients	10 Clients	3 Clients	5 Clients	10 Clients
Local	0.7357 (0.0030)	0.7325 (0.0066)	0.8039 (0.0008)	0.6674 (0.0069)	0.6647 (0.0045)	0.7128 (0.0035)	0.8445 (0.0003)	0.8108 (0.0000)	0.8024 (0.0011)	0.6724 (0.0003)	0.7959 (0.0106)	0.7562 (0.0137)	0.6523 (0.0221)	0.5764 (0.0001)	0.6645 (0.0051)
FedAvg	0.8416 (0.0044)	0.6332 (0.0166)	0.7162 (0.0382)	0.7426 (0.0024)	0.7498 (0.0049)	0.7252 (0.0035)	0.7126 (0.0000)	0.8640 (0.0024)	0.8586 (0.0010)	0.7668 (0.0414)	0.5695 (0.0483)	0.5669 (0.0974)	0.5626 (0.0715)	0.4195 (0.0173)	0.4858 (0.0187)
FedSAGE+	0.7560 (0.0237)	0.4156 (0.0034)	0.3522 (0.1196)	0.7505 (0.0150)	0.5167 (0.0389)	0.4929 (0.0075)	0.8980 (0.0001)	0.9091 (0.0025)	0.9041 (0.0012)	0.9239 (0.0083)	0.6670 (0.0206)	0.6246 (0.0585)	0.7539 (0.0062)	0.6934 (0.0006)	0.6656 (0.0082)
FedGCN	0.8226 (0.0062)	0.8124 (0.0158)	0.7243 (0.0172)	0.7376 (0.0111)	0.7649 (0.0010)	0.7123 (0.0122)	0.7127 (0.0000)	0.8504 (0.0011)	0.8441 (0.0070)	0.7398 (0.0036)	0.5717 (0.0583)	0.5627 (0.0957)	0.5782 (0.0623)	0.4217 (0.0243)	0.4908 (0.0183)
FedPUB	0.8476 (0.0021)	0.8448 (0.0009)	**0.8622** (0.0059)	0.7455 (0.0065)	0.7694 (0.0074)	0.7505 (0.0081)	0.9064 (0.0016)	0.9069 (0.0019)	0.9092 (0.0019)	0.9399 (0.0020)	0.9122 (0.0016)	0.8983 (0.0052)	0.8339 (0.0142)	0.8202 (0.0141)	0.8181 (0.0124)
FedNTD	0.8452 (0.0067)	0.8526 (0.0024)	0.6984 (0.0030)	0.7455 (0.0069)	**0.7826** (0.0047)	0.7146 (0.0079)	0.9049 (0.0002)	0.9065 (0.0009)	0.9061 (0.0012)	0.9378 (0.0029)	0.9166 (0.0021)	0.9119 (0.0036)	0.8492 (0.0107)	0.8619 (0.0034)	0.8707 (0.0055)
FedED	0.8542 (0.0084)	0.8398 (0.0024)	0.6779 (0.0343)	0.7305 (0.0086)	0.7624 (0.0050)	0.6251 (0.0149)	0.9080 (0.0006)	0.9086 (0.0027)	0.8985 (0.0025)	0.9463 (0.0014)	0.9101 (0.0027)	0.8950 (0.0059)	0.8623 (0.0136)	0.8722 (0.0035)	0.8356 (0.0158)
FedLoG	**0.8601** (0.0118)	**0.8575** (0.0074)	0.8451 (0.0103)	**0.7663** (0.0086)	0.7728 (0.0049)	**0.7624** (0.0063)	**0.9180** (0.0005)	**0.9129** (0.0015)	**0.9115** (0.0043)	**0.9653** (0.0020)	**0.9496** (0.0037)	**0.9305** (0.0049)	**0.9073** (0.0012)	**0.8986** (0.0014)	**0.8742** (0.0107)

(b) Missing Class

	Cora			CiteSeer			PubMed			Amazon Photo			Amazon Computers		
Methods	3 Clients	5 Clients	10 Clients	3 Clients	5 Clients	10 Clients	3 Clients	5 Clients	10 Clients	3 Clients	5 Clients	10 Clients	3 Clients	5 Clients	10 Clients
Local	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)
FedAvg	0.3900 (0.1104)	0.1119 (0.0202)	0.0652 (0.0568)	0.2022 (0.0751)	0.1914 (0.0140)	0.3189 (0.0218)	0.0000 (0.0000)	0.0013 (0.0013)	0.0020 (0.0010)	0.0000 (0.0000)	0.0000 (0.0000)	0.0085 (0.0148)	0.0000 (0.0000)	0.0000 (0.0000)	0.0073 (0.0127)
FedSAGE+	0.5000 (0.0457)	0.1393 (0.0317)	0.0287 (0.0111)	0.5581 (0.0524)	0.1622 (0.0470)	0.3701 (0.0528)	0.0000 (0.0000)	0.0015 (0.0013)	0.0034 (0.0004)	0.0000 (0.0000)	0.0000 (0.0000)	0.0036 (0.0051)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)
FedGCN	0.0702 (0.0713)	0.2123 (0.0197)	0.0549 (0.0091)	0.1648 (0.0187)	0.1702 (0.0833)	0.0833 (0.0584)	0.0000 (0.0000)	0.0000 (0.0000)	0.0006 (0.0006)	0.0000 (0.0000)	0.0156 (0.0271)	0.0097 (0.0169)	0.0000 (0.0000)	0.0000 (0.0000)	0.0085 (0.0148)
FedPUB	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0012 (0.0021)	0.0053 (0.0026)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0002 (0.0003)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)
FedNTD	0.3714 (0.1273)	0.1895 (0.0098)	0.0336 (0.0317)	0.4257 (0.0077)	0.2438 (0.0476)	0.2878 (0.0459)	0.0003 (0.0002)	0.0256 (0.0256)	0.0512 (0.0050)	0.0000 (0.0000)	0.0019 (0.0009)	0.0061 (0.0064)	0.0038 (0.0015)	0.0008 (0.0008)	0.0104 (0.0024)
FedED	0.5305 (0.1078)	0.1080 (0.0158)	0.0350 (0.0305)	0.3184 (0.2660)	0.1534 (0.0470)	0.1039 (0.1785)	0.0056 (0.0045)	0.0097 (0.0097)	0.0050 (0.0022)	0.2192 (0.0395)	0.1796 (0.0634)	0.0004 (0.0007)	0.1162 (0.0297)	0.0005 (0.0005)	0.0004 (0.0007)
FedLoG	**0.6472** (0.0811)	**0.4948** (0.0930)	**0.4037** (0.0619)	**0.6142** (0.0292)	**0.5922** (0.1037)	**0.5958** (0.0608)	**0.1070** (0.0623)	**0.1700** (0.1236)	**0.2290** (0.0308)	**0.4795** (0.0949)	**0.5525** (0.1464)	**0.1328** (0.0562)	**0.3580** (0.1256)	**0.2175** (0.0638)	**0.0412** (0.0327)

r using the weighted average $\phi^{(r+1)} \leftarrow \frac{1}{|\mathcal{V}|}\sum_{k=1}^{K}|\mathcal{V}_k|\phi_k^{(r)}$, where $|\mathcal{V}|$ is the total number of nodes, and $|\mathcal{V}_k|$ is the number of nodes for the k-th client.

Global Synthetic Data Generation. In addition, the server generates global synthetic data, which will be employed during the Local Generalization phase (Section 3.3) to help mitigate the issue of local overfitting. More specifically, we first generate node features in global synthetic data $\mathcal{D}_g$ by merging the head degree condensed nodes $\mathcal{V}_{k,\text{head}}$ from all clients, weighted by the proportion of head classes for each client, since each client has expert knowledge of the dominant classes within their data as shown in Appendix C. More formally, for each class $c \in C$, the feature vector of the i-th global synthetic node for class c, $x_{v_g^{(c,i)}}$, is generated as follows:

$$x_{v_g^{(c,i)}} = \frac{1}{\sum_{k=1}^{K} r_k^c} \sum_{k=1}^{K} r_k^c x_{v_{k,\text{head}}^{(c,i)}}, \tag{7}$$

where $r_k^c = \frac{|\mathcal{V}_k^c|}{|\mathcal{V}_k|}$ represents the proportion of nodes labeled c in the k-th client's dataset. In this way, the server generates $s \times |C_{\mathcal{V}}|$ global synthetic nodes $\mathcal{V}_g$ with features

$$X_{\mathcal{V}_g} = \{x_{v_g^{(1,1)}}, \ldots, x_{v_g^{(1,s)}}, \ldots, x_{v_g^{(|C_{\mathcal{V}}|,1)}}, \ldots, x_{v_g^{(|C_{\mathcal{V}}|,s)}}\},$$

encompassing knowledge related to both head degree and head class across clients. Thus, the initially generated global synthetic data is $\mathcal{D}_g = (\mathcal{G}_g = (\mathcal{V}_g, \mathcal{E}_\varnothing), Y_{\mathcal{V}_g})$ with node features $X_{\mathcal{V}_g}$.

Although structural information is vital for understanding the data distribution in graph-structured data, the initially generated global synthetic data $\mathcal{D}_g$ comprises only the feature information of nodes and lacks any surrounding neighbor information. Consequently, we propose generating a single neighbor node that can effectively represent the h-hop subgraph of the synthetic node, using pre-trained neighbor generators. These generators are specifically trained to produce class-specific neighbors, as elaborated in Appendix B. Local clients utilize these neighbor generators at the local level to adaptively customize the features of the global synthetic data, and then generate the neighbors of the customized features, as detailed in following Section 3.3.

In summary, at the end of each round r, the server distributes the model weights $\phi^{(r+1)}$ and the global synthetic data $\mathcal{D}_g$.

3.3 Local Generalization

At the beginning of each round ($r > 1$), each client initializes its local model with the distributed global model parameters ϕ. After the local update within their local data (i.e., Local Fitting), we additionally train the local model with the global synthetic data $\mathcal{D}_g$, enabling it to generalize to locally absent knowledge (i.e., Local Generalization), such as tail and missing classes. Since each client has different locally absent knowledge, we first adaptively customize the global synthetic data for the current state of local model and then train with the customized data.

Local Adaptation – Feature Scaling. Motivated by recent work [1], we adjust the difficulty of the training data class-wise to prevent overfitting to major classes while allowing the model to effectively learn tail and missing classes. To achieve this, we use feature scaling on the global synthetic data $X_{\mathcal{V}_g}$ as follows:

$$\hat{x}_{v_g^{(c,i)}} = x_{v_g^{(c,i)}} + \gamma_k[c] \cdot (\bar{x}_{\mathcal{V}_g} - x_{v_g^{(c,i)}}), \tag{8}$$

where $\bar{x}_{\mathcal{V}_g} = \frac{1}{|\mathcal{V}_g|}\sum_{x_{v_g} \in X_{\mathcal{V}_g}} x_{v_g}$ and $\gamma_k \in \mathbb{R}^{|C_{\mathcal{V}}|}$ is a class-wise adaptive factor that adjusts prediction difficulty by moving the global synthetic data of class c to the average of global synthetic data, making it harder to predict. Note that when $\gamma_k = 1$, the global synthetic data for class c is completely replaced with the average of the global synthetic data, whereas the global synthetic data for class c remains unchanged when $\gamma_k = 0$. During training, we dynamically modify the factor by incrementing it by 0.001 whenever the local model's accuracy for class c exceeds the threshold at the end of the round, thereby increasing the difficulty of the corresponding class.

Local Adaptation – Neighbor Generation. As structural information is crucial for optimizing the GNN model, we generate a neighbor for each customized global synthetic data using the class-specific neighbor generator $\mathbf{NG}^c$ for the corresponding class c, as shown in Figure 1(c). We then construct the synthetic graph set $\mathcal{G}_k^{\text{syn}}$, which consists of graphs, each containing the customized global synthetic node neighboring with its generated neighbor node. It is worth noting that the generated neighbor node effectively represents the h-hop subgraph structure of each synthetic node. The details of how to pre-train the neighbor generators are provided in Appendix B.

Train with Customized Data. Using the synthetic graphs $\mathcal{G}_k^{\text{syn}}$, the prediction for the target synthetic nodes within $\mathcal{V}_g$ follows Eqs. 1-5 using the same local model ϕ_k as for normal nodes $\mathcal{V}_k$, with α set to 0.5. The prediction loss for $\mathcal{D}_g$ is $\mathcal{L}_{k,g} = \sum_{v_g \in \mathcal{V}_g} \sum_{c \in C_{\mathcal{V}}} -\mathbb{I}(y_{v_g} = c) \log(\mathbf{p}[c])$. At the end of each round, we adjust the adaptive factor γ_k based on the class prediction accuracy.

In summary, the final loss of the local model is $\mathcal{L}_k = \mathcal{L}_{k,cls} + \mathcal{L}_{k,g}$.

4 EXPERIMENTS

4.1 Experimental Settings

Datasets. We conduct experiments on five real-world graph datasets. Distributed subgraphs are constructed by dividing each dataset into a certain number of clients using the METIS graph partitioning algorithm [7]. The datasets used are Cora, CiteSeer, PubMed [14], Amazon Computer, and Amazon Photo [12, 15]. For more details, see Appendix J.

Baseline Methods. 1) Local: Refers to local training without any weight sharing. **2) FedAvg** [13]: The most widely-used FL baseline. **3) FedSAGE+** [21], **4) FedGCN** [18] and **5) FedPUB** [2]: subgraph-FL baselines that primarily address missing knowledge within the current local label distribution. To ensure a fair comparison, we also evaluate our method against **6) FedNTD** [9] and **7) FedED** [4], which address local overfitting in FL. For more details, see Appendix K.

Evaluation Protocol. We perform FL for 100 rounds. Node classification accuracy is measured on the client side and averaged across all clients over three runs. More details are in Appendix L.

4.2 Experiment Results

Q1. How FedLoG perform in conventional FL settings? Table 1(a) presents the evaluation of models on graphs that were used for training. The label distributions of the test nodes match the training label distribution of each client. We refer to this conventional setting as **Seen Graph**, where models are evaluated on test nodes within the same graph structure as the training nodes (i.e., transductive setting [8]). The overall performance of FedLoG on the 'Seen Graph' outperforms that of other baselines, demonstrating its strong performance in conventional settings.

Q2. Does FedLoG generalize to unseen data after local updates? In this section, we introduce a practical test setting for subgraph-FL, evaluating on unseen node with missing classes within their local graphs (i.e., **Missing Class** setting) to measure the model's generalization performance on potential future unseen data. (Details of other possible scenarios of unseen data, such as 'Unseen node with seen classes' and 'New client never participated in the training phase', are provided in Appendix G.1). Specifically, each client has new nodes with missing classes added to its local graph. In order to predict unseen nodes with missing classes, extensive knowledge from other clients is required. We evaluate the performance on new nodes representing missing classes for each client, assessing how effectively the FL framework enables the local model to learn previously absent knowledge.

In Table 1(b), Local and personalized FL models like FedPUB [2] fail to predict the missing classes as they optimize for the training label distribution. Although FedSAGE+ [21] and FedGCN [18] attempt to compensate for missing neighbors, they are not always effective because the missing class is not always within the neighbors. Moreover, FedNTD [9] and FedED [4] address local overfitting and achieve relatively high performance in missing class prediction. However, they regularize the local model logits to match the global model, risking noisy information from tail data and resulting in inconsistent performance across different settings.

In contrast, FedLoG alleviates local overfitting by using reliable class representations and structural information across clients, reducing the emphasis on noisy information. Thus, FedLoG successfully addresses unseen data, ensuring robust performance even with missing classes due to its generalization ability across all classes and structural features.

Additional experiments, including ablation studies and hyperparameter analysis, are provided in Appendix G. In addition, we provide the privacy analysis of the synthetic data and the communication overhead in Appendix H and Appendix I, respectively.

5 CONCLUSION

In this study, we address the challenges of local overfitting and unseen nodes in subgraph-FL with our proposed method, FedLoG. Our model generates global synthetic data by condensing reliable information from each class representation and its structural information across clients, enabling adaptive generalization of absent knowledge within local datasets. This approach enhances the generalization capabilities of local models, allowing them to handle unseen data effectively. Our experimental results demonstrate that FedLoG outperforms existing baselines, proving its efficacy in practical scenarios for generalizing to unseen data.

REFERENCES

[1] Sumyeong Ahn, Jongwoo Ko, and Se-Young Yun. 2023. Cuda: Curriculum of data augmentation for long-tailed recognition. *arXiv preprint arXiv:2302.05499* (2023).
[2] Jinheon Baek, Wonyong Jeong, Jiongdao Jin, Jaehong Yoon, and Sung Ju Hwang. 2023. Personalized subgraph federated learning. In *International Conference on Machine Learning*. PMLR.
[3] Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge Belongie. 2019. Class-balanced loss based on effective number of samples. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition.*
[4] Kuangpu Guo, Yuhe Ding, Jian Liang, Ran He, Zilei Wang, and Tieniu Tan. 2024. Not all Minorities are Equal: Empty-Class-Aware Distillation for Heterogeneous Federated Learning. *arXiv preprint arXiv:2401.02329* (2024).
[5] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).
[6] Haibo He and Yunqian Ma. 2013. Imbalanced learning: foundations, algorithms, and applications. (2013).
[7] George Karypis and Vipin Kumar. 1997. METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. (1997).
[8] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).
[9] Gihun Lee, Minchan Jeong, Yongjin Shin, Sangmin Bae, and Se-Young Yun. 2022. Preservation of the global knowledge by not-true distillation in federated learning. *Advances in Neural Information Processing Systems* 35 (2022).
[10] Rui Liu, Pengwei Xing, Zichao Deng, Anran Li, Cuntai Guan, and Han Yu. 2024. Federated Graph Neural Networks: Overview, Techniques, and Challenges. *IEEE Transactions on Neural Networks and Learning Systems* (2024).
[11] Zemin Liu, Trung-Kien Nguyen, and Yuan Fang. 2021. Tail-gnn: Tail-node graph neural networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge*

Discovery & Data Mining.

[12] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. 2015. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*.

[13] 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.

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

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

[16] Arjun Subramonian, Jian Kang, and Yizhou Sun. 2024. Theoretical and Empirical Insights into the Origins of Degree Bias in Graph Neural Networks. *arXiv preprint arXiv:2404.03139* (2024).

[17] Chuhan Wu, Fangzhao Wu, Yang Cao, Yongfeng Huang, and Xing Xie. 2021. Fedgnn: Federated graph neural network for privacy-preserving recommendation. *arXiv preprint arXiv:2102.04925* (2021).

[18] Yuhang Yao, Weizhao Jin, Srivatsan Ravi, and Carlee Joe-Wong. 2024. FedGCN: Convergence-Communication Tradeoffs in Federated Training of Graph Convolutional Networks. *Advances in Neural Information Processing Systems* 36 (2024).

[19] Sukwon Yun, Kibum Kim, Kanghoon Yoon, and Chanyoung Park. 2022. Lte4g: Long-tail experts for graph neural networks. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*.

[20] Jie Zhang, Zhiqi Li, Bo Li, Jianghe Xu, Shuang Wu, Shouhong Ding, and Chao Wu. 2022. Federated learning with label distribution skew via logits calibration. In *International Conference on Machine Learning*. PMLR.

[21] Ke Zhang, Carl Yang, Xiaoxiao Li, Lichao Sun, and Siu Ming Yiu. 2021. Subgraph federated learning with missing neighbor generation. *Advances in Neural Information Processing Systems* 34 (2021).

[22] Tianxiang Zhao, Xiang Zhang, and Suhang Wang. 2021. Graphsmote: Imbalanced node classification on graphs with graph neural networks. In *Proceedings of the 14th ACM international conference on web search and data mining*.

Supplementary Material for Subgraph Federated Learning for Local Generalization

A RELATED WORKS

A.1 Subgraph Federated Learning

Recent works [10] have introduced FL frameworks that enable collaborative GNN training without sharing graph data. Subgraph-FL aims to leverage disjoint graphs from each local client to collaboratively train a global model for solving downstream tasks. Existing studies [10, 17, 18, 21] have attempted to supplement the local absent knowledge among local graphs that each client currently holds. For instance, FedSAGE+ [21], FedGNN [17], and FedGCN [18] request node information from other clients to recover missing neighborhood nodes and compensate for potential edges. FedPUB [2] focuses on personalizing the local model by finding similar communities with the current local data across the clients. However, due to the mutable properties of graph domains, subgraph-FL must generalize well not only to the current label distribution but also to new nodes that will emerge in the future. Unlike these approaches [2, 17, 18, 21] that only focus on finding missing knowledge relevant to the current state, our model learns representations for all classes and their connection patterns, ensuring better generalization across various future scenarios.

A.2 Local Overfitting in Federated Learning

Imbalanced data distribution is common in real-world scenarios, and significant efforts [3, 19] have been made to address the resulting deterioration in model performance. Federated Learning cannot avoid the data imbalance problem, as the presence of multiple clients implies that each client has its own data imbalance, making it prone to overfitting to the local data [4, 9, 20]. Recent works [4, 9, 20] aim to alleviate local overfitting in FL by regularizing local models to be similar to the global model. FedLC [20], and FedED [4] introduce logit calibration, which aligns the logits of each class in the local models more closely with those of the global model. While FedED [4] addresses the missing class problem in FL, it does not consider the noisy properties of tail data [16]. Our method, FedLoG, addresses local overfitting and ensures reliable representation of all classes by leveraging class-specific knowledge across clients and considering their structural properties. To the best of our knowledge, this is the first work to tackle local overfitting with missing classes in subgraph-FL.

B DETAILED PROCESS OF PRETRAINING THE NEIGHBOR GENERATOR

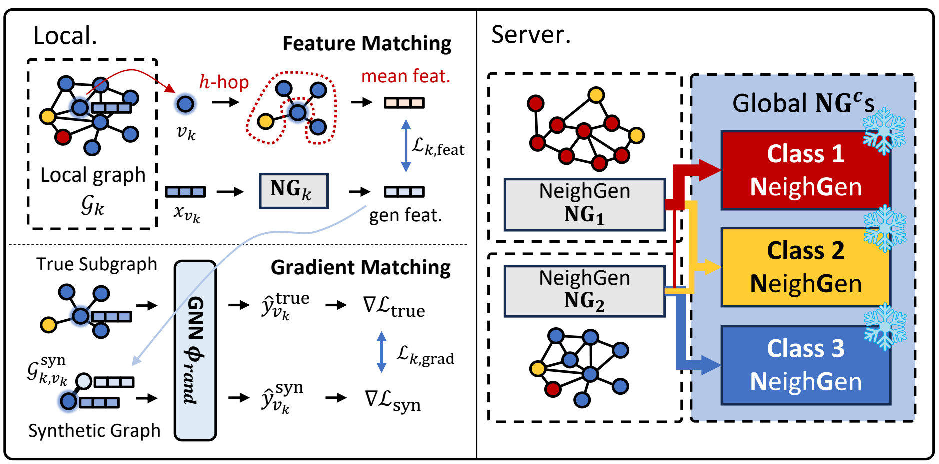

Figure 2: Overview of Pretraining the Neighbor Generator.

In this section, we will conceptually outline the process of pretraining local neighbor generators and the method of aggregating these generators on the server to produce unbiased local neighbors.

Local. In Figure 2, we train a neighbor generator $\mathbf{NG}_k$ for each local client k. The input to the neighbor generator is the feature vector of the target node v_k, denoted by x_{v_k}. The neighbor generator produces a feature vector $\hat{x}_{\mathcal{N}_{v_k}} = \mathbf{NG}_k(x_{v_k}) \in \mathbb{R}^{1\times d}$, where $\mathcal{N}_{v_k}$ is a single generated neighbor node for the target node v_k. Then we generate a subgraph for the target node, which we denote as $\mathcal{G}^{\text{syn}}_{k,v_k} = (\mathcal{V}_{v_k} = \{v_k, \mathcal{N}_{v_k}\}, \mathcal{E}_{k,v_k})$, where $\mathcal{E}_{k,v_k}$ has only one connection between nodes v_k and $\mathcal{N}_{v_k}$.

This process condenses information from the h-hop subgraph around v_k into the generated neighbor node and mimics the training effect of the true h-hop subgraph into the generated subgraph $\mathcal{G}^{\text{syn}}_{k,v_k}$. To achieve this, for any target node $v_k \in \mathcal{V}_k$, we extract the h-hop subgraph $\mathcal{G}_{k,v_k} = (\mathcal{V}_{k,v_k}, \mathcal{E}_{k,v_k}) \subseteq \mathcal{G}_k$, then minimize the distance between the average features of real neighbors and generated features:

$$\mathcal{L}_{k,\text{feat}} = \frac{1}{|\mathcal{V}_k|}\sum_{v_k\in\mathcal{V}_k}\|\hat{x}_{\mathcal{N}_{v_k}} - x_{\mathcal{N}_{v_k}}\|_2^2, \text{ where } x_{\mathcal{N}_{v_k}} = \frac{1}{|\mathcal{V}_{k,v_k}|-1}\sum_{v\in\mathcal{V}_{k,v_k}\setminus v_k} x_v. \quad (9)$$

After that, we minimize the difference between gradients of GNNs applied to $\mathcal{G}^{\text{syn}}_{k,v_k}$ and $\mathcal{G}_{k,v_k}$ with N randomly initialized weights $\phi^{(n)}_{\text{rand}}$ (for $n = 1, 2, \ldots, N$) optimizing for the target-node classification task as follows:

$$\mathcal{L}_{k,\text{grad}} = \frac{1}{N}\sum_{n=1}^{N}\left\|\nabla_{\phi^{(n)}_{\text{rand}}} l(\phi^{(n)}_{\text{rand}}; \mathcal{G}^{\text{syn}}_{k,v_k}(v_k), y_{v_k}) - \nabla_{\phi^{(n)}_{\text{rand}}} l(\phi^{(n)}_{\text{rand}}; \mathcal{G}_{k,v_k}(v_k), y_{v_k})\right\|_2^2 \tag{10}$$

The final loss for optimizing the local neighbor generator is defined as

$$\mathcal{L}_{\text{NG}_k} = \mathcal{L}_{k,\text{feat}} + \mathcal{L}_{k,\text{grad}}. \tag{11}$$

We pretrain $\mathbf{NG}_k$ for all $k \in \{1, \ldots, K\}$ over P epochs (i.e., 100) using the training sets within each local dataset, resulting in a collection $\mathfrak{N} = \{\mathbf{NG}_1, \ldots, \mathbf{NG}_K\}$. We set N as 20.

Server. Since clients have different label distributions, local neighbor generators tend to be biased towards generating neighbors of the dominant class within their respective local graphs. This means that each local neighbor generator has expert knowledge of generating neighbors for dominant classes within its local data. Therefore, as shown in Figure 1(a), we aggregate the local neighbor generators on a class-wise basis, weighting them by the proportion of each class within the local training set.

Formally, for each class $c \in C$, the aggregated neighbor generator $\mathbf{NG}^c$ is defined as:

$$\mathbf{NG}^c = \frac{1}{\sum_{k=1}^{K} r_k^c}\sum_{k=1}^{K} r_k^c \mathbf{NG}_k, \tag{12}$$

where $r_k^c = \frac{|\mathcal{V}_k^c|}{|\mathcal{V}_k|}$ represents the proportion of nodes with label c in the k-th client's local dataset, $\mathcal{V}_k^c$ is the set of nodes with label c in the k-th client's graph, and $|\mathcal{V}_k|$ is the total number of nodes in the k-th client's graph. In practice, we only utilize the nodes within the training set for counting the number of nodes. Consequently, we generate class-specific neighbor generators $\mathbf{NG}^c$ for all $c \in C$. These generators are then frozen and shared with all clients for the entire federated learning process.

C ANALYSIS OF RELIABILITY OF HEAD DEGREE AND HEAD CLASS NODES

In this section, we detail the process of evaluating data reliability within the graph data. We define "Data Reliability" as the accuracy and consistency of information from decentralized nodes. Specifically, we assess which data within the local dataset positively or negatively impacts other clients in the FL framework. Inspired by the robust performance of GNNs on head class and head degree nodes [19, 22], we found that data reliability largely depends on **1)** the extent of data connections (i.e., degree headness) and **2)** the predominance of certain classes (i.e., class headness).

We set the base settings for both perspectives. In the FL framework, we assign two roles to each client. The 'Receiver' is the client who receives information about the target class from other clients. This client is trained using the same training data across all settings for this section, ensuring a fair comparison to validate the impact from other clients. 'Contributors' are the clients who share knowledge from their own data with the 'Receiver'. Their training sets (i.e., information shared through the FL framework) vary for each setting, such as adjusting the proportion of head/tail degree nodes or class imbalance rate. In a global setting with K clients in FL, we assign one client as the 'Receiver' and the others as 'Contributors' (i.e., $K-1$ clients).

To assess how degree or class headness affects data reliability, we measure the target class accuracy of the 'Receiver' when varying the training sets of 'Contributors'. This helps identify whether headness or tailness of data positively or negatively impacts the 'Receiver'. We construct the global model by averaging the weights from each client and then evaluate the global model on the 'Receiver's' local graph following FedAvg [13].

Impact of Degree Headness on the Data Reliability. We divide head degree and tail degree using the tail degree threshold λ set to 3, as justified in Appendix E. Nodes with degrees less than or equal to 3 are considered tail degree nodes, while those with degrees greater than 3 are head degree nodes. We only vary the training dataset of the 'Contributors'. We create three different training sets for each 'Contributor': 1) Head Degree nodes only (Head degree), 2) Tail Degree nodes only (Tail degree), and 3) Balanced degree nodes (Head+Tail degree). Each training set contains the same number of nodes, but their headness differs according to the setting. The 'Head degree' setting includes only head degree nodes with the target class, the 'Tail degree' setting includes only tail degree nodes, and the 'Balanced degree' setting includes an equal mix of head and tail degree nodes. We use the FedAvg [13] framework for the federated learning setting with 100 rounds. At the final round, we evaluate the accuracy of the target class within the 'Receiver's' local data using the global model. We average the performances across all classes and report the mean of three seeds results.

Figure 3(left) shows receiver accuracy with contributor training sets composed of **1)** 'Head Degree nodes only (Head Deg.)', **2)** 'Tail Degree nodes only (Tail degree)', and **3)** 'Balanced degree nodes (Head+Tail Deg.)'. The receiver's performance improves with knowledge from head degree nodes within contributors, indicating their reliability over tail degree nodes. As the number of clients increases, the performance gap widens, highlighting the negative impact of noise within tail degree nodes.

Impact of Class Headness on the Data Reliability. We define the 'Imbalance Rate' using the proportion of the number of the target class within each 'Contributor"s local data. We fix the training nodes of the target class for each 'Contibutor', and varies the number of training nodes for other classes which are not the target class. Let n_c be the number of nodes per class, and let the number of training nodes of the target class be n_t. We then assign n_k number of training nodes for each non-target class:

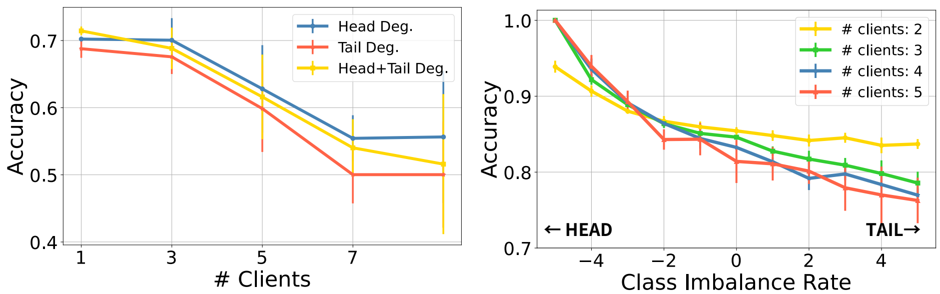

Figure 3: Data Reliability Analysis (PubMed used).

$$n_k = n_t + \frac{r_{\text{imb}}}{10} \times \min(n_c \forall c \in C) \tag{13}$$

where $\min(n_c \forall c \in C)$ is the minimum number of nodes across all other classes c within the set C, and r_{imb} is the imbalance rate can be defined as:

$$r_{\text{imb}} = 10 \times \frac{n_k - n_t}{\min(n_c \forall c \in C)}.$$

Thus, if the r_{imb} has a negative value (i.e., $n_t > n_k$), it means the target class becomes a head class within the local data. Conversely, when the r_{imb} has a positive value, the target class becomes a tail class. As the value of r_{imb} increases, the tailness of the target class gets higher. We set r_{imb} in the range from -5 to +5, and we average the performances of the 'Receiver' at the final round across all classes and report the mean of three seed results.

Figure 3(right) manipulates label distribution within each contributor's local graph, transforming the target class into a tail or head class by varying the number of training nodes in other classes while keeping the target class constant. When the class headness of the target class within contributors is high (i.e., negative imbalance rates), contributors enhance the receiver's performance. Conversely, with low headness (i.e., positive imbalance rates), the receiver's performance deteriorates as contributors struggle to represent the target class [19]. This negative impact is magnified with more clients.

In summary, data with 'headness' in both degree and class from other clients helps a client learn reliable representations, while 'tailness' data negatively impacts learning due to insufficient or noisy information. Our method, FedLoG, collects knowledge from head degree and head class data across all clients to alleviate locally absent knowledge.

D DETAILED PROCESS OF THE CLASSIFIER $\varphi_{k,H}$ AND $\varphi_{k,T}$

In this section, we provide the detailed process of the classifiers $\varphi_{k,H}$ and $\varphi_{k,T}$. Specifically, we describe the details of Eq. 4:

$$h'_{v_k} = \varphi_{k,H}(h_{v_k}, \{h_{v_k} - h_{v^{(0,1)}_{k,\text{head}}}, \ldots, h_{v_k} - h_{v^{(|C_\mathcal{V}|,s)}_{k,\text{head}}}\}),$$

which is one of the classifiers (i.e., the classifier for the head degree branch) within the local model. Since both the head degree and tail degree classifiers have the same architecture, we describe the details only for the head degree classifier.

Our main purpose for the classifier is to ensure that all prototypes P_{head} participate in the final prediction of the target node, so the prediction loss is influenced by all prototypes. For this, we first generate a message $\mathbf{m}_{kj}$ from a prototype node $v_j \in P_{k,\text{head}}$ to the target node v_k using the distance d_{kj} between them utilizing MLP_{msg}:

$$\mathbf{m}_{kj} = \text{MLP}_{\text{msg}}([h_{v_k} \parallel \mathbf{n}_{v_k}, d_{kj}]), \tag{14}$$

where d_{kj} is calculated as:

$$d_{kj} = \|\Delta \mathbf{r}_{kj}\|^2, \quad \Delta \mathbf{r}_{kj} = [h_{v_k} \parallel \mathbf{n}_{v_k}] - [h_{v_j} \parallel h_{v_j}], \tag{15}$$

and $\parallel$ denotes concatenation. Here, $\mathbf{n}_{v_k} = \frac{1}{|\mathcal{N}_{v_k}|} \sum_{v_o \in \mathcal{N}_{v_k}} h_{v_o}$ represents the average embedding of the 1-hop neighbors of the target node v_k. We use neighbor information because the representations of neighbors from high-degree nodes and low-degree nodes differ, allowing each branch to leverage degree-specific knowledge. The target node embedding h_{v_k} is then updated by applying a learned transformation to the representation differences $h_{v_k} - h_{v_j}$ and aggregating these transformations:

$$\mathbf{t}_{kj} = (h_{v_k} - h_{v_j}) \odot \text{MLP}_{\text{trans}}(\mathbf{m}_{kj}), \tag{16}$$

$$\mathbf{T}_k = \frac{1}{|P_{k,\text{head}}|} \sum_j \mathbf{t}_{kj}, \tag{17}$$

$$h'_{v_k} = h_{v_k} + \mathbf{T}_k, \tag{18}$$

where $\text{MLP}_{\text{trans}}$ transform the message m_{kj} into a scalar value.

In summary, the classifiers $\varphi_{k,H}$ and $\varphi_{k,T}$ update the embedding of the target node h_i by reflecting the interaction between all different prototypes in P_{head} so that the final prediction and its loss are influenced by all prototypes (i.e., learnable synthetic nodes).

E CRITERIA FOR THRESHOLD DEGREE VALUE FOR TAIL-DEGREE NODES

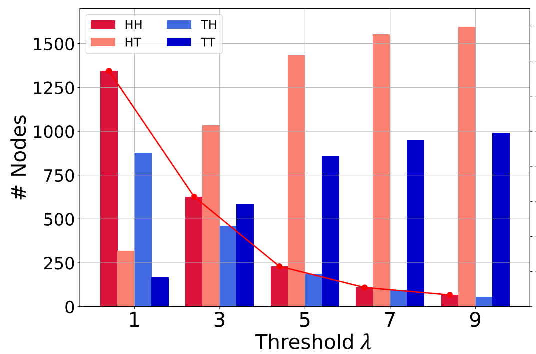

Figure 4: The number of nodes for HH/HT/TH/TT at threshold λ (Cora dataset used).

Recent methods [11, 19] addressing the degree long-tail problem consider nodes with degrees less than or equal to 5 as tail degree nodes, while those with degrees greater than 5 are considered head degree nodes.

As shown in Figure 4, we illustrate the number of nodes belonging to 1) Head class & Head degree (HH), 2) Head class & Tail degree (HT), 3) Tail class & Head degree (TH), and 4) Tail class & Tail degree (TT) as we vary the threshold value λ within the global graph. We use the Cora dataset for validation.

When the threshold λ increases, the number of HH nodes significantly decreases, reducing the amount of knowledge that can be condensed into the global synthetic data. In this work, we set λ to 3 to utilize a sufficient amount of HH knowledge while filtering out noisy information from tail degree nodes.

F DETAILED PROCESS OF EVALUATING UNSEEN DATA

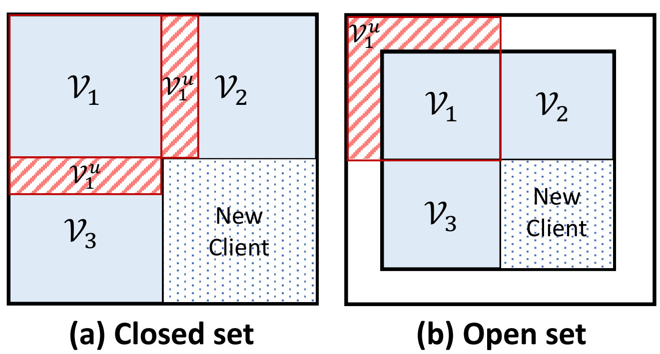

Figure 5: Overview of Unseen Data settings ($K = 3$).

In this section, we provide a detailed description of our proposed 'Unseen Data' test settings (i.e., 'Unseen Node', 'Missing Class', and 'New Client'). To evaluate realistic scenarios, we define two different settings for evaluating unseen data: **1)** Closed set nodes setting (Closed set) and **2)** Open set nodes setting (Open set). The results in Table 1 are evaluated on the closed set nodes setting.

F.1 Closed Set

Following recent work [2], we partition the global graph into several subgraphs using the Metis graph partitioning algorithm [7]. For the 'New Client' setting, we generate an additional subgraph, resulting in the partitioning of the global graph into $k + 1$ subgraphs, where k denotes the number of clients. Due to the properties of the Metis algorithm, the extra subgraph has a distinct label distribution, as the algorithm minimizes the number of edges between partitions, leading to the formation of distinct communities.

Table 2: Model performance across settings. Mean accuracy with standard deviation over 3 runs.

(a) Unseen Node

	Cora			CiteSeer			PubMed			Amazon Photo			Amazon Computers		
Methods	3 Clients	5 Clients	10 Clients	3 Clients	5 Clients	10 Clients	3 Clients	5 Clients	10 Clients	3 Clients	5 Clients	10 Clients	3 Clients	5 Clients	10 Clients
Local	0.1250 (0.0030)	0.2957 (0.0079)	0.2854 (0.0263)	0.4443 (0.0131)	0.3471 (0.0020)	0.5177 (0.0052)	0.7510 (0.0010)	0.7292 (0.0000)	0.7489 (0.0013)	0.1333 (0.0000)	0.1900 (0.0392)	0.3958 (0.0211)	0.1687 (0.0000)	0.2488 (0.0000)	0.3890 (0.0043)
FedAvg	0.5403 (0.0797)	0.5198 (0.0179)	0.4139 (0.1308)	0.6585 (0.0220)	0.6098 (0.0301)	0.6199 (0.0084)	0.6154 (0.0000)	0.8189 (0.0120)	0.8070 (0.0043)	0.1782 (0.0419)	0.2125 (0.0136)	0.3727 (0.0221)	0.2275 (0.1017)	0.3095 (0.0179)	0.4177 (0.0335)
FedSAGE+	0.5653 (0.0546)	0.4265 (0.0062)	0.3836 (0.0705)	0.6572 (0.0093)	0.4023 (0.0339)	0.6154 (0.0034)	0.8944 (0.0048)	0.8921 (0.0089)	0.8926 (0.0051)	0.4781 (0.0093)	0.2298 (0.0394)	0.4607 (0.0391)	0.3462 (0.0325)	0.3555 (0.0196)	0.3596 (0.0024)
FedGCN	0.3689 (0.0646)	0.5877 (0.0018)	0.5075 (0.0001)	0.6232 (0.0243)	0.6530 (0.1095)	0.6139 (0.0119)	0.6154 (0.0000)	0.7759 (0.0058)	0.8188 (0.0049)	0.2565 (0.0067)	0.2604 (0.0077)	0.3708 (0.0304)	0.2086 (0.0597)	0.3084 (0.0140)	0.4103 (0.0342)
FedPUB	0.5529 (0.0246)	0.5192 (0.0064)	0.4767 (0.0286)	0.6798 (0.0334)	0.6691 (0.0057)	0.6938 (0.0245)	0.8878 (0.0003)	0.8822 (0.0056)	0.8836 (0.0043)	0.4085 (0.0118)	0.3890 (0.0404)	0.5033 (0.0155)	0.4414 (0.0225)	0.5025 (0.0466)	0.5253 (0.0471)
FedNTD	0.6355 (0.0195)	0.5880 (0.0041)	0.3913 (0.1235)	0.7057 (0.0173)	0.7014 (0.0614)	0.6151 (0.0155)	0.8939 (0.0068)	0.8852 (0.0044)	0.8816 (0.0031)	0.4042 (0.0155)	0.5833 (0.0043)	0.5286 (0.0030)	0.5056 (0.0440)	0.6034 (0.0309)	0.6482 (0.0226)
FedED	0.7338 (0.0294)	0.5514 (0.0117)	0.3916 (0.1184)	0.6646 (0.0658)	0.6148 (0.0097)	0.5381 (0.0781)	0.9008 (0.0027)	0.8884 (0.0036)	0.8730 (0.0077)	0.6227 (0.0429)	0.4265 (0.0675)	0.4629 (0.0137)	0.4582 (0.0176)	0.5408 (0.0223)	0.4940 (0.0275)
FedLoG	**0.7341** (0.0273)	**0.7413** (0.0316)	**0.7406** (0.0527)	**0.7624** (0.0522)	**0.7415** (0.0142)	**0.8044** (0.0078)	**0.9044** (0.0021)	**0.8956** (0.0033)	**0.8965** (0.0061)	**0.7065** (0.0715)	**0.7077** (0.0571)	**0.7176** (0.0277)	**0.7677** (0.0237)	**0.8156** (0.0326)	**0.6735** (0.0292)

(b) New Client

	Cora			CiteSeer			PubMed			Amazon Photo			Amazon Computers		
Methods	3 Clients	5 Clients	10 Clients	3 Clients	5 Clients	10 Clients	3 Clients	5 Clients	10 Clients	3 Clients	5 Clients	10 Clients	3 Clients	5 Clients	10 Clients
Local	0.0995 (0.0084)	0.1488 (0.0059)	0.1778 (0.0284)	0.1435 (0.0113)	0.1968 (0.0006)	0.1337 (0.0000)	0.3570 (0.0000)	0.3947 (0.0000)	0.3936 (0.0001)	0.0313 (0.0001)	0.0965 (0.0004)	0.1169 (0.0089)	0.1571 (0.0000)	0.1974 (0.0000)	0.2816 (0.0111)
FedAvg	0.3583 (0.0206)	0.2713 (0.0057)	0.3924 (0.1880)	0.2572 (0.0222)	0.2859 (0.0348)	0.2976 (0.0138)	0.3333 (0.0000)	0.5243 (0.0138)	0.5175 (0.0240)	0.2597 (0.0118)	0.0853 (0.0044)	0.1661 (0.0172)	0.1978 (0.0000)	0.2924 (0.0637)	0.4799 (0.0904)
FedSAGE+	0.2411 (0.0109)	0.3250 (0.0226)	0.4129 (0.1052)	0.3630 (0.0385)	0.0646 (0.0099)	0.1048 (0.0322)	0.3834 (0.0022)	0.5616 (0.0126)	0.5279 (0.0000)	0.2782 (0.0086)	0.0900 (0.0021)	0.1472 (0.0549)	0.2132 (0.3683)	0.3718 (0.1044)	0.2756 (0.0062)
FedGCN	0.3449 (0.0494)	0.3320 (0.0052)	0.4825 (0.0189)	0.2572 (0.0072)	0.4548 (0.1900)	0.2144 (0.0566)	0.3333 (0.0000)	0.4830 (0.0048)	0.4890 (0.0067)	0.2597 (0.0550)	0.0659 (0.0226)	0.1605 (0.0169)	0.2066 (0.0781)	0.3009 (0.0590)	0.4841 (0.0920)
FedPUB	0.3990 (0.0239)	0.2258 (0.0153)	0.4031 (0.0087)	0.3929 (0.0485)	0.3408 (0.0113)	0.3930 (0.0296)	0.4112 (0.0007)	0.6036 (0.0060)	0.5743 (0.0008)	0.3171 (0.0120)	0.1075 (0.0102)	0.2540 (0.0078)	0.5244 (0.0276)	0.6764 (0.0289)	0.5543 (0.0586)
FedNTD	0.3805 (0.0328)	0.3169 (0.0010)	0.3705 (0.1879)	0.4321 (0.0479)	0.5288 (0.0940)	0.3057 (0.0166)	0.4153 (0.0039)	0.6321 (0.0093)	0.6026 (0.0106)	0.4617 (0.1349)	0.1473 (0.0036)	0.2980 (0.0361)	0.0038 (0.0602)	0.7146 (0.0101)	0.7873 (0.0232)
FedED	0.4527 (0.0353)	0.2537 (0.0165)	0.3194 (0.1364)	0.3303 (0.1068)	0.4053 (0.0240)	0.1346 (0.0136)	0.4842 (0.0226)	0.6352 (0.0019)	0.5969 (0.0131)	0.5451 (0.0586)	0.1563 (0.0098)	0.2622 (0.0027)	0.1162 (0.0712)	0.7147 (0.0067)	0.5228 (0.0610)
FedLoG	**0.5047** (0.0884)	**0.4439** (0.0455)	**0.6055** (0.0914)	**0.5973** (0.1623)	**0.5647** (0.0179)	**0.6487** (0.1143)	**0.6053** (0.1293)	**0.7091** (0.0557)	**0.7546** (0.0107)	**0.5605** (0.1052)	**0.5083** (0.1794)	**0.5574** (0.0368)	**0.7386** (0.0190)	**0.9164** (0.0071)	**0.8029** (0.0866)

The closed set setting includes unseen data for the 'Unseen Node' and 'Missing Class' settings from other clients. Specifically, in Figure 5(a), the global set of nodes is $\mathcal{V} = \bigcup_{k=1}^{K} \mathcal{V}_k$, with $\mathcal{V}_i \cap \mathcal{V}_j = \varnothing$ for all $i \neq j$. We construct the 'Unseen Node' and 'Missing Class' nodes for client k by expanding the h-hop subgraph from the local graph $\mathcal{G}_k$ at testing time. Since we allocate all nodes within the global node set $\mathcal{V}$ to the clients, the nodes within the h-hop subgraph (i.e., $\mathcal{V}_k^u$) inevitably overlap with those of other clients. Although nodes may overlap, no edges are shared between different clients. Unseen nodes from other clients establish new connections with the local data.

For the 'Missing Class' setting, we select the missing classes for each client and then exclude the nodes corresponding to those classes (i.e., $\mathcal{V}_k^{\text{uc}}$) within each local graph $\mathcal{G}_k$. To maintain the overall context of the local graph, we select the missing class from tail classes, which have the smallest portion within each local graph. If the number of nodes corresponding to the missing classes is insufficient, we add additional missing classes for those clients. Excluded nodes $\mathcal{V}_k^{\text{uc}}$ are included in $\mathcal{V}_k^u$.

When evaluating the 'Missing Class' at test time, we expand the local graph $\mathcal{G}_k$ to the range of h-hop, and within the evolved graph structure, the local model predicts the labels of nodes in $\mathcal{V}_k^{\text{uc}}$. For 'Unseen Node', the local model predicts the labels of nodes in $\mathcal{V}_k^u \setminus \mathcal{V}_k^{\text{uc}}$.

For real-world case for the Closed Set setting, consider Store-A, which uses a model tailored to the purchasing habits of its regular customers. This model may struggle to adapt to the distinct buying patterns of customers from Store-B. These new patterns could create unfamiliar 'also-bought' connections between products within Store-A, especially if they involve new products that Store-A has never sold before. However, these customers can visit Store-A at any time, forming new relationships with existing nodes, reflecting a real-world scenario. This complexity increases the difficulty in effectively integrating and addressing new nodes in the model. In addition, we provide the data statistics for each setting in Appendix O.

F.2 Open Set

In real-world scenarios, unseen data outside the global nodes $\mathcal{V}$ in the FL system can emerge and form new relationships with existing nodes. We define this setting as Open Set, where the unseen nodes are $\mathcal{V}_k^u \cap \mathcal{V} = \varnothing$. To create this setting, we randomly crop 20% of the global graph before partitioning it into $k+1$ subgraphs, denoting the cropped node set as $\mathcal{V}_{\text{crop}}$.

Similar to the Closed Set, we exclude nodes corresponding to locally assigned missing classes within each local graph. At test time, for the 'Unseen Node' and 'Missing Class' settings, we reconstruct the structure between cropped nodes $\mathcal{V}_{\text{crop}}$ and local nodes $\mathcal{V}_k$. Within the reconstructed graph, we evaluate the nodes in $\mathcal{V}_{\text{crop}}$ that belong to the missing classes for the 'Missing Class' setting and those having locally trained classes for the 'Unseen Node' setting.

In Table 3, we provide the experimental results on the open set in Appendix G.5.

G ADDITIONAL EXPERIMENTS

G.1 Performance on Unseen Node & New Client settings

In Table 2, we provide the results of performance on 'Unseen Node' and 'New Client' settings of baselines and our proposed method FedLoG. Detailed settings are described in Appendix F.

1) Unseen Node (Table 2(a)). Each client has new nodes with seen classes added to its local graph, which introduce structural changes. We perform evaluations on the new nodes to assess how well the FL framework adapts to these structural changes.

2) New Client (Table 2(b)). A new client that has never participated in the FL framework emerges. This client has a distinct label distribution and graph structure. We assess how well the FL framework generalizes to accommodate this new client, ensuring robust performance across diverse scenarios. We evaluate the nodes within the unseen graph using each model after local updates, and then report the mean accuracy for all clients.

In these experiments, our proposed method,FedLoG, demonstrates superior performance compared to other models. Specifically, in the 'Unseen Node' setting, FedLoG shows a remarkable ability to adapt to new structural changes introduced by the addition of new nodes with seen classes. This adaptability is crucial for maintaining model accuracy and reliability in dynamic environments where the structure of the data can change over time.

In the 'New Client' scenario, FedLoG's performance is particularly noteworthy. The method shows excellent generalization capabilities, effectively handling the introduction of a new client with a unique label distribution and graph structure. This scenario simulates real-world situations where new clients with different data characteristics join the FL framework. FedLoG's ability to maintain high performance in this setting highlights its robustness and flexibility, making it a reliable choice for diverse and evolving federated learning environments.

Overall, FedLoG consistently outperforms other models in both settings, showcasing its effectiveness in adapting to new data and generalizing across different scenarios. These results underscore the potential of FedLoG as a powerful tool for federated learning applications, where adaptability and generalization are critical for success.

G.2 Do the headness of degree and class really help other clients?

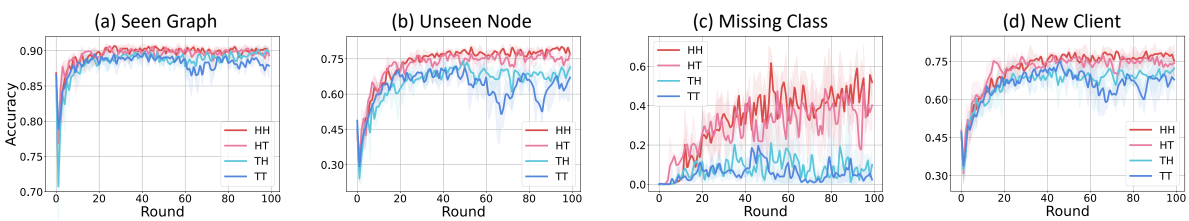

Figure 6: Impact of headness of class/degree for various scenarios (Amazon Clothing - 3 Clients).

We evaluate the importance of the headness of class/degree under various scenarios, both of which are expected to enhance the data reliability. As FedLoG additionally trains the clients using global synthetic data, we measure the impact by varying the knowledge condensed into the global synthetic data. Specifically, we compare four different test settings for constructing global synthetic data using **1)** Head Class & Head Degree nodes (HH), **2)** Head Class & Tail Degree nodes (HT), **3)** Tail Class & Head Degree nodes (TH), and **4)** Tail Class & Tail Degree nodes (TT). Detailed descriptions are provided in Appendix M.

Figure 6 shows test accuracy curves that verify the impact of each test setting on performance and stability. Data reliability varies with global synthetic data knowledge, with HH knowledge being the most reliable. Class headness significantly affects reliability, evident in the performance gap between head and tail classes. Degree headness impacts stability, with tail degree settings showing more fluctuations. Thus, we can conclude that using HH knowledge is crucial for maintaining reliability and stable outcomes.

G.3 Ablation Study

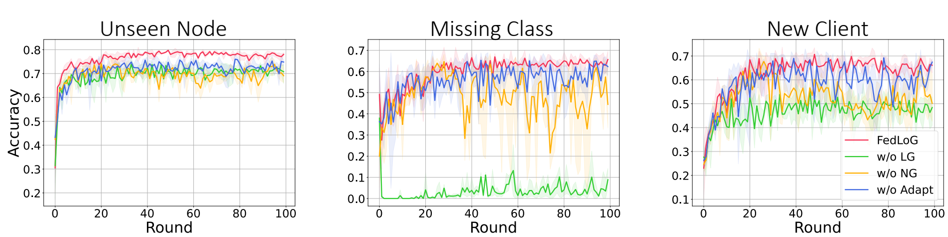

Figure 7: Ablation studies (CiteSeer - 3 Clients).

We perform an ablation study on **1)** Local Generalization (w/o LG), **2)** Neighbor Generator (w/o NG), and **3)** Adaptive Factor (w/o Adapt). As these modules are all directly related to addressing unseen data, we depict the test accuracy curves in Unseen Data settings to easily verify the effectiveness of each module.

Local Generalization. Local Generalization is an essential phase to prevent local overfitting after the local updates of each client within the FL framework. The Local Generalization phase enables clients to learn locally absent knowledge from the global synthetic data, allowing them to generalize all classes even if they don't have any data for certain classes within their local data (i.e., missing class). As shown in Figure 7, our method without the Local Generalization phase fails to generalize the missing class, which means Local Generalization is crucial for addressing the absent knowledge. Furthermore, for the Unseen Node and New Client settings, the performance deteriorates when we omit the Local Generalization phase.

Neighbor Generator. We evaluate the effectiveness of the neighbor generators $\mathbf{NG}^c \forall [c]$. The neighbor generators generate the neighbors of the global synthetic data $\mathcal{D}_g$, which contain the h-hop neighbor information for the target nodes and also contribute to training by mimicking the true h-hop subgraphs' gradient. We perform the ablation study for the neighbor generators by omitting the generation of neighbors for the global synthetic data, which means we train them without any generated neighbors. In Figure 7, without neighbors, there is a discrepancy in the training mechanism of the GNN between isolated nodes and nodes within the graph structure, leading to a performance decrease for all settings. Furthermore, the learning curves fluctuate when training the global synthetic data without neighbor generation, indicating that using only the features of synthetic nodes negatively affects stability.

Adaptive Factor γ. The Adaptive Factor γ helps each client learn all classes adaptively. The Adaptive Factor adjusts the difficulty of the global synthetic data for each client depending on the class prediction ability for all classes at the current round. Thus, the Adaptive Factor affects the stability of learning for each client. In Figure 7, we can verify the effectiveness of the Adaptive Factor, as the learning curves are more fluctuating than the original FedLoG method, and the performance is decreased.

G.4 Impact of the Hyperparameters

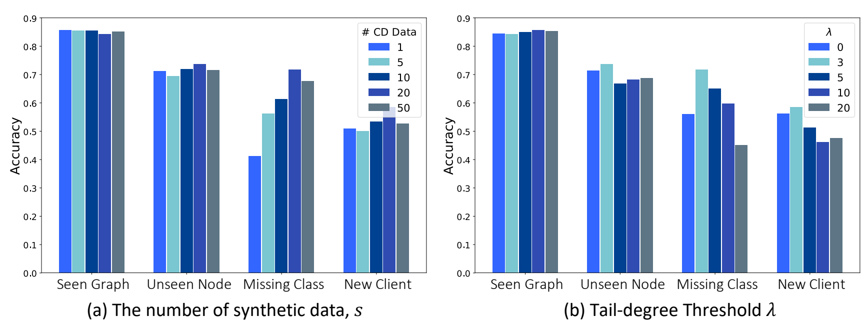

Figure 8: Hyperparameter analysis.

In Figure 8, we analyze the impact of hyperparameters such as the number of synthetic data for each class (s) and the tail degree threshold (λ).

The Number of Synthetic Data, s. For generating the global synthetic data, sets of learnable nodes $\mathcal{V}_{k,\text{head}}$ and $\mathcal{V}_{k,\text{tail}}$ are constructed during the Local Fitting phase within each client. We assign s learnable synthetic nodes per class and vary s to assess its impact on global synthetic nodes.

As shown in Figure 8(a), we vary s within the range $[1, 5, 10, 20, 50]$ and evaluate the model's performance on the same test data using the Cora dataset with 3 clients. Notably, s significantly impacts the 'Unseen Data' settings, particularly the 'Missing Class' setting, which relies heavily on global synthetic data. A larger number of synthetic data condenses diverse knowledge expressions. However, too many synthetic data points complicate modeling the interaction between the target node and each synthetic nodes (i.e., prototypes), as all prototypes participate in the final prediction described in Section 3.1. Consequently, accuracy for 'Unseen Data'—including 'Unseen Node', 'Missing Class', and 'New Client'—improves with more synthetic data, but an excessive number (e.g., $s = 50$) can reduce performance. Conversely, the performance of the 'Seen Graph' settings shows robustness to the number of synthetic data compared to the 'Unseen Data' settings because the dependency on knowledge from other clients is lower for test data with the same distribution as the training data.

Tail-Degree Threshold λ. We evaluate the impact of the tail-degree threshold λ on performance. Varying λ within the range $[0, 3, 5, 10, 20]$, we use the CiteSeer dataset with 3 clients for the evaluation. As shown in Figure 8(b), the tail-degree threshold λ significantly impacts the 'Unseen Data' settings as it directly influences the knowledge condensed into the global synthetic data. Increasing λ filters out more knowledge from tail-degree nodes, condensing primarily head-degree node knowledge. However, as illustrated in Figure 4 in Section E, the number of HH nodes significantly decreases with a higher λ, reducing the amount of knowledge to be condensed into the global synthetic

Table 3: Performance on Unseen Node and Missing Class in the Open Set setting.

(a) Unseen Node

	Cora			CiteSeer			PubMed			Amazon Photo			Amazon Computers		
Methods	3 Clients	5 Clients	10 Clients	3 Clients	5 Clients	10 Clients	3 Clients	5 Clients	10 Clients	3 Clients	5 Clients	10 Clients	3 Clients	5 Clients	10 Clients
Local	0.1250 (0.0030)	0.2957 (0.0077)	0.2854 (0.0263)	0.4443 (0.0131)	0.3471 (0.0020)	0.5177 (0.0052)	0.7510 (0.0010)	0.7292 (0.0000)	0.7489 (0.0013)	0.1333 (0.0000)	0.1900 (0.0039)	0.3958 (0.0211)	0.1687 (0.0001)	0.2891 (0.0000)	0.3890 (0.0043)
FedAvg	0.6696 (0.0232)	0.5939 (0.0215)	0.4243 (0.1304)	0.6055 (0.0033)	0.7126 (0.0210)	0.5255 (0.0119)	0.8679 (0.0059)	0.7192 (0.0170)	0.6793 (0.0127)	0.2481 (0.0455)	0.2491 (0.0671)	0.2692 (0.0304)	0.3480 (0.0428)	0.2980 (0.0198)	0.2617 (0.0074)
FedSAGE+	0.6362 (0.0764)	0.5050 (0.0047)	0.3953 (0.0527)	0.4090 (0.0155)	0.2667 (0.0160)	0.3945 (0.0571)	**0.9035** (0.0028)	0.8820 (0.0015)	0.8312 (0.0124)	0.3117 (0.0071)	0.2529 (0.0198)	0.3651 (0.0183)	0.4205 (0.0073)	0.6028 (0.0055)	0.3404 (0.0189)
FedGCN	0.6840 (0.0083)	0.6299 (0.0022)	0.4389 (0.1433)	0.6148 (0.0124)	0.6500 (0.0319)	0.5767 (0.0254)	0.8571 (0.0027)	0.7138 (0.0114)	0.6558 (0.0044)	0.2329 (0.0448)	0.2411 (0.0512)	0.2617 (0.0238)	0.3519 (0.0506)	0.2923 (0.0235)	0.2621 (0.0041)
FedPUB	0.6772 (0.0039)	0.5971 (0.0117)	0.4717 (0.0114)	0.6097 (0.0264)	0.7222 (0.0087)	0.5958 (0.0049)	0.8842 (0.0114)	**0.8954** (0.0022)	**0.8864** (0.0023)	0.4842 (0.0204)	0.5109 (0.0331)	0.3790 (0.0359)	0.4886 (0.0123)	0.5068 (0.0286)	0.4574 (0.0125)
FedNTD	0.7066 (0.0241)	0.6402 (0.0076)	0.4245 (0.1274)	0.6443 (0.0105)	0.7639 (0.0082)	0.5664 (0.0168)	0.8953 (0.0052)	0.8769 (0.0033)	0.8789 (0.0013)	0.5516 (0.0283)	0.6196 (0.0098)	0.4903 (0.0165)	0.4183 (0.0033)	0.6707 (0.0252)	0.6778 (0.0102)
FedED	0.6904 (0.0163)	0.5453 (0.0185)	0.3024 (0.0038)	0.5985 (0.0330)	0.6568 (0.0060)	0.4448 (0.0232)	0.8978 (0.0047)	0.8771 (0.0043)	0.8805 (0.0028)	0.6491 (0.0346)	0.5872 (0.0395)	0.2581 (0.0460)	0.4326 (0.0119)	0.7420 (0.0291)	0.5751 (0.0374)
FedLoG	**0.7224** (0.0102)	**0.7163** (0.0216)	**0.6203** (0.0089)	**0.6363** (0.0153)	**0.7645** (0.0141)	**0.6634** (0.0235)	0.8627 (0.0078)	0.8622 (0.0058)	0.8627 (0.0062)	**0.8754** (0.0049)	**0.8275** (0.0340)	**0.6576** (0.0202)	**0.7759** (0.0475)	**0.8625** (0.0180)	**0.7163** (0.0279)

(b) Missing Class

	Cora			CiteSeer			PubMed			Amazon Photo			Amazon Computers		
Methods	3 Clients	5 Clients	10 Clients	3 Clients	5 Clients	10 Clients	3 Clients	5 Clients	10 Clients	3 Clients	5 Clients	10 Clients	3 Clients	5 Clients	10 Clients
Local	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0001 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)
FedAvg	0.0000 (0.0000)	0.2091 (0.0291)	0.0000 (0.0317)	0.1801 (0.0405)	0.4269 (0.0517)	0.1490 (0.0387)	0.2771 (0.0207)	0.0499 (0.0133)	0.0166 (0.0064)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0246)
FedSAGE+	0.2030 (0.0883)	0.2774 (0.0528)	0.0244 (0.0326)	0.3243 (0.2832)	0.4155 (0.0220)	0.2007 (0.1161)	0.0495 (0.0116)	0.0733 (0.0160)	0.1166 (0.0217)	0.0000 (0.0000)	0.0000 (0.0000)	0.0175 (0.0304)	0.0000 (0.0000)	0.0000 (0.0000)	0.0189 (0.0327)
FedGCN	0.0000 (0.0000)	0.2940 (0.0280)	0.0579 (0.0520)	0.0961 (0.0364)	0.3562 (0.1246)	0.1831 (0.0253)	0.2035 (0.0165)	0.0478 (0.0058)	0.0049 (0.0012)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0177 (0.0085)
FedPUB	0.0000 (0.0000)	0.0082 (0.0094)	0.0352 (0.0000)	0.0000 (0.0000)	0.0251 (0.0220)	0.0070 (0.0060)	0.0318 (0.0125)	0.0002 (0.0004)	0.0100 (0.0087)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)	0.0000 (0.0000)
FedNTD	0.1182 (0.0686)	0.2650 (0.0189)	0.0457 (0.0402)	0.4054 (0.0375)	0.5822 (0.0504)	0.2054 (0.0166)	0.0733 (0.0638)	0.2688 (0.0659)	0.3895 (0.0263)	0.0292 (0.0479)	0.0385 (0.0376)	0.1558 (0.0305)	0.1708 (0.0137)	0.0088 (0.0047)	0.0017 (0.0014)
FedED	0.1333 (0.0844)	0.1449 (0.0143)	0.0370 (0.0339)	0.2523 (0.0364)	0.2922 (0.0412)	0.1197 (0.0219)	0.1487 (0.0460)	0.1118 (0.0696)	0.1604 (0.0067)	0.0503 (0.0820)	0.0000 (0.0000)	0.0029 (0.0050)	0.0613 (0.0434)	0.1232 (0.0973)	0.0091 (0.0063)
FedLoG	**0.4273** (0.0567)	**0.5528** (0.0569)	**0.2649** (0.0174)	**0.4234** (0.0324)	**0.5342** (0.0449)	**0.4484** (0.0203)	**0.5697** (0.2118)	**0.4758** (0.1292)	**0.5697** (0.2169)	**0.3333** (0.0142)	**0.5423** (0.1803)	**0.4397** (0.1365)	**0.7648** (0.0985)	**0.2548** (0.0204)	**0.2929** (0.0348)

data. Thus, setting λ to 3 yields the best performance, effectively filtering out tail-degree knowledge while ensuring a sufficient amount of HH nodes.

G.5 Experimental Results on the Open Set

We evaluate the Unseen Node and Missing Class in the Open Set settings to validate the model's ability to generalize to nodes never seen at the global level. The results are provided in Table 3. Similar to the Closed Set, our method, FedLoG, outperforms the baselines across most settings. However, in the Unseen Node setting on the PubMed dataset, some baselines show better performance than our method. We attribute this to the PubMed dataset providing a sufficient number of training data for each class, allowing methods to generalize well within each class's local data. Conversely, in the Missing Class setting, the baselines fail to generalize due to the absence of local data for the missing classes. In contrast, our model effectively generalizes to all classes, including missing classes, demonstrating its robustness on various real-world scenarios.

H PRIVACY ANALYSIS

In this section, we analyze the potential privacy problem with FedLoG.

1) Does utilizing the class distribution of the clients pose a privacy problem? The class distribution does not include individual data but merely represents the proportion for each class. This information is less sensitive than raw data. Class distribution is general statistical information that indicates the trends within a group rather than specific data about individual users. Therefore, it is very difficult for an attacker to infer specific data of individual nodes from the class distribution.

If privacy issues arise from knowing the trends of the group, we can add noise to the class rate to make it difficult to determine the exact class distribution. We experimented with two methods of adding noise to the class rate: **1)** adding class-wise Gaussian noise with μ as 0 and σ as $a \times r_k^c$, where a is in the range $[0.01, 0.1, 0.2, 0.5]$ and $r_k^c = \frac{|\mathcal{V}_k^c|}{|\mathcal{V}_k|}$, and **2)** performing random permutation of the elements in the class rate vector. To maintain the trend while applying random permutation, we permuted only the elements within the head classes and within the tail classes.

Table 4: GN: Gaussian Noise, RP: Random Permutation (Cora dataset with 3 clients used)

	FedLoG w.o. noise	FedLoG w. GN (a = 0.01)	FedLoG w. GN (a = 0.1)	FedLoG w. GN (a = 0.2)	FedLoG w. GN (a = 0.5)	FedLoG w. RP
Seen Graph	**0.8601** (0.0118)	**0.8530** (0.0089)	**0.8542** (0.0080)	**0.8583** (0.0054)	**0.8560** (0.0010)	**0.8631** (0.0010)
Unseen Node	**0.7341** (0.0273)	**0.7127** (0.0191)	**0.7217** (0.0318)	**0.7336** (0.0252)	**0.7057** (0.0123)	**0.7351** (0.0054)
Missing Class	**0.6472** (0.0811)	**0.6244** (0.05275)	**0.6032** (0.0457)	**0.6540** (0.0967)	**0.6277** (0.0148)	**0.6328** (0.0586)
New Client	**0.5047** (0.0884)	**0.5278** (0.0326)	**0.5297** (0.0523)	**0.5401** (0.0658)	**0.4883** (0.0232)	**0.5199** (0.0421)

As shown in Table 4, both Gaussian Noise (GN) and Random Permutation (RP) noise methods, which are roughly similar to the original class rate, showed no significant difference in performance, except for the GN (a = 0.5) setting that highly deteriorates the trend of the class rate, indicating that our model does not require an exact class distribution as long as the general trend is maintained. Through this approach, we can protect privacy more rigorously.

2) Can synthetic data be specified to match the original data's features? We acknowledge that since the synthetic data is generated by condensing the original nodes within each client's graph, there may be a potential risk to privacy. One possible way to validate the violation of privacy is to examine the difference in the feature distribution of the original nodes (i.e., $\mathcal{V}_k$) and that of the synthetic nodes (i.e., $\mathcal{V}_{k,\text{head}}$ and $\mathcal{V}_{k,\text{tail}}$). That is, if the two distributions overlap, then we can say that the original node features can be reconstructed from the synthetic data, which indicates the potential risk of privacy violation. In Figure 9, we visualize the 2-dimensional PCA of both the original feature matrix (blue) and the synthetic feature matrix (orange) for the same class (using the CiteSeer dataset). We observe that the PCA visualization of these two matrices is significantly different, indicating that sharing the synthetic nodes poses less risk of privacy violation.

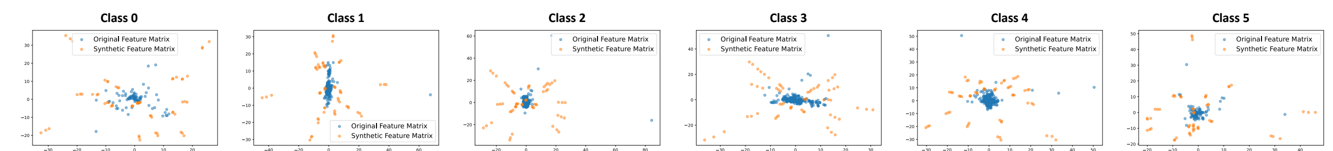

Figure 9: 2-dimensional PCA visualization of feature distributions for the same class in the CiteSeer dataset.

Additionally, the risk of privacy violation is further alleviated as synthetic data cannot be assigned to individual nodes, since it contains features aggregated from all training nodes. This aggregation also condenses the structural information into the features, leading to a different distribution compared to the original feature space, which does not contain the structural information. as the synthetic nodes have no inherent structure. This is because the synthetic nodes have no structure compared to the original nodes. Therefore, the structural information from the original nodes must be distilled into the synthetic node's feature space, resulting in a different distribution from the original feature space.

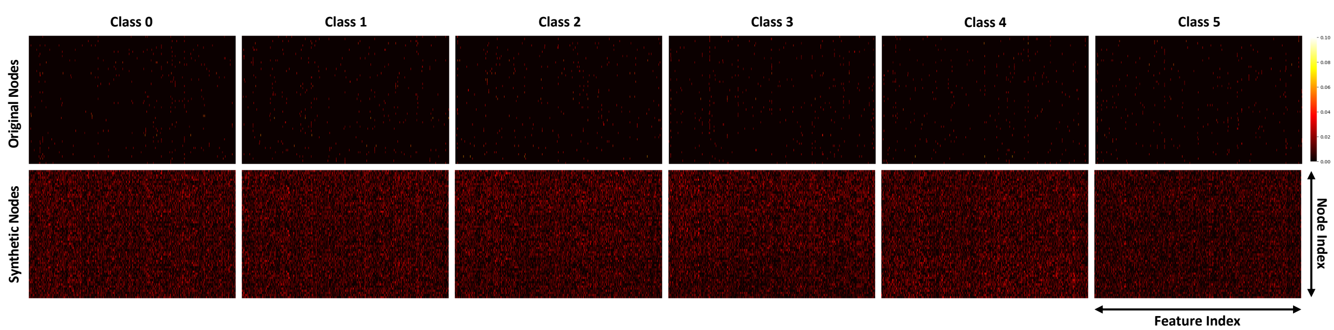

Figure 10: Heatmap visualization of feature distributions for the CiteSeer dataset.

Moreover, in Figure 10, we visualize the feature distribution of both the original features (top row) and the condensed synthetic features (bottom row) for the same class in a heatmap. As shown, the values of the features within each feature set show significant differences, making it difficult to reconstruct the raw features of the corresponding class in the original data.

3) Protection Against Gradient Inversion Attacks. In the context of adversaries attempting to restore original data from gradients uploaded to the server (i.e., gradient inversion attacks), FedLoG provides enhanced protection. This protection is achieved because each local client is trained not only on local data but also on global synthetic data. The inclusion of global synthetic data introduces noise into the gradients from the adversaries' perspective, making it harder for them to extract information solely from the original data.

4) Privacy Enhancement Through Feature Scaling Variability. Moreover, each client applies different extents of feature scaling (Section 3.3) to the global synthetic data, and these scaling extents are never shared with others. This variation further complicates the task of distinguishing the gradients originating from the global synthetic data. As a result, it becomes more challenging for adversaries to accurately invert the gradients and reconstruct the original data.

I COMMUNICATION OVERHEAD

We provide comparisons of communication overhead across different baselines. FedLoG uploads and downloads both the synthetic data and the model parameters at the end of each round. For the Cora dataset with a setting of 3 clients and $s = 20$, our model has 1,081,926 parameters to share with the server, resulting in 4 bytes×1,081,926=4.32 MB (excluding the neighbor generator, which is only trained once at the first round). Additionally, the synthetic data has 182,000 parameters ($s \times |C_{\mathcal{V}}| \times d$, where $|C_{\mathcal{V}}|$ denotes the number of classes and d denotes the dimension of the features), amounting to 0.72 MB. In summary, our model requires 10.08 MB ($2 \times (4.32 + 0.72)$) for upload and download each round. Below are comparisons of communication overhead between models over 100 rounds:

	FedAvg	FedSAGE+	FedGCN	FedPUB	FedNTD	FedED	FedLoG
Cost (MB)	393.11	1543.58	393.11	786.03	393.11	393.11	1011.14

Table 5: Comparison of Communication Cost (MB) Across Different Models

Although FedLoG relatively requires higher communication overhead compared to other baselines, it shows faster convergence due to its utilization of reliable class representation, leading to a stable training process. Below are comparisons of communication overhead until each model reaches the same accuracy (i.e., 0.8 on the Cora dataset with 3 clients).

Model	FedAvg	FedSAGE+	FedGCN	FedPUB	FedNTD	FedED	FedLoG
Rounds to Reach 0.8	58	100 (Fails to reach)	57	29	19	39	10
Cost (MB)	228.00	1543.58	224.07	227.95	72.79	149.41	101.11

Table 6: Comparison of Rounds to Reach 0.8 Accuracy and Communication Cost Across Different Models

Despite FedLoG's higher communication overhead per round, its faster convergence results in a lower overall communication overhead to achieve the same accuracy compared to other baselines. This demonstrates the efficiency and stability of FedLoG's training process, making it an effective approach despite the initially higher communication cost per round.

J DATASETS

Cora [14]: The Cora dataset consists of 2,708 scientific publications classified into one of seven classes. The citation network contains 5,429 links. Each publication in the dataset is described by a 1,433-dimensional binary vector, indicating the absence/presence of a word from a dictionary.

CiteSeer [14]: The CiteSeer dataset comprises 3,327 scientific publications classified into one of six classes. The citation network consists of 4,732 links. Each publication is described by a 3,703-dimensional binary vector.

PubMed [14]: The PubMed dataset includes 19,717 scientific publications from the PubMed database pertaining to diabetes, classified into one of three classes. The citation network comprises 44,338 links. Each publication is described by a TF/IDF-weighted word vector from a dictionary with a size of 500.

Amazon Computers [12]: The Amazon Computers dataset is a subset of the Amazon co-purchase graph. It consists of 13,752 nodes (products) and 245,861 edges (co-purchase relationships). Each product is described by a 767-dimensional feature vector, and the task is to classify products into 10 classes.

Amazon Photos [15]: The Amazon Photos dataset is another subset of the Amazon co-purchase graph. It consists of 7,650 nodes (products) and 143,663 edges (co-purchase relationships). Each product is described by a 745-dimensional feature vector, and the task is to classify products into 8 classes.

K BASELINES

In this section, we provide implementation details for the baselines.

Local. This is a non-FL baseline where each local model is trained independently using the GCN embedder without any weight sharing.

Table 7: Dataset Statistics

Dataset	Nodes	Edges	Features	Classes	Description
Cora	2,708	5,429	1,433	7	Scientific publications
CiteSeer	3,327	4,732	3,703	6	Scientific publications
PubMed	19,717	44,338	500	3	Scientific publications
Amazon Computers	13,752	245,861	767	10	Amazon co-purchase
Amazon Photos	7,650	143,663	745	8	Amazon co-purchase

FedAvg. [13]. This FL baseline involves clients sending their local model weights to the server, which then averages these weights based on the number of training samples at each client. The aggregated model is then distributed back to the clients. In our implementation, we use GCN as the graph embedder.

FedSAGE+. [21]. This subgraph-FL baseline involves clients using GraphSAGE as an embedder and a missing neighbor generator, trained using a graph mending technique. The neighbor generator creates missing neighbors based on their number and features. With the neighbor generator, local models are trained with compensated neighbors and then their weights are aggregated on the server using FedAvg-based FL aggregation.

FedGCN. [18]. This subgraph-FL baseline involves clients who collect h-hop averaged neighbor node features from other clients at the beginning of training to address missing information. The server then collects local model weights for FedAvg-based FL aggregation.

FedPUB. [2]. This subgraph-FL baseline proposes weight aggregation based on the similarity between clients. It identifies highly correlated clients with similar community graph structures by using the functional embeddings of local GNNs, which are computed using random graphs as inputs to determine similarities.

FedNTD. [9]. This FL baseline is designed to tackle the challenge of overfitting in local models due to non-IID data across clients. It performs local-side distillation only for non-true classes to prevent forgetting global knowledge corresponding to regions outside the local distribution. In our implementation, we use GCN as the graph embedder.

FedED. [4]. This FL baseline is designed to tackle the challenge of overfitting in local models and addresses the issue of local missing classes. Similar to our task, it addresses the missing class problem in FL by adding a loss term that regularizes the logits of missing classes to be similar to those of the global model. In our implementation, we use GCN as the graph embedder.

L IMPLEMENTATION DETAILS

In this section, we provide implementation details of FedLoG.

Model Architecture. In our experiments, we use a 2-layer GraphSAGE [5] implementation (φ_E) with a dropout rate of 0.5, a hidden dimension of 128, and an output dimension of 64. The model parameters with learnable features $X_{\mathcal{V}_{\mathrm{k,head}}}$ and $X_{\mathcal{V}_{k,\mathrm{tail}}}$ are optimized with Adam optimizer using a learning rate of 0.001. The classifiers φ_H and φ_T consist of 2 main learnable functions (i.e., $\mathbf{MLP}_{\mathrm{msg}}$ and $\mathbf{MLP}_{\mathrm{trans}}$) as follows:

- **Message generating function** ($\mathbf{MLP}_{\mathrm{msg}}$): Two linear layers with SiLU activation (Inputs $\rightarrow$ Linear ($2 \times 64 \rightarrow 64$) $\rightarrow$ SiLU $\rightarrow$ Linear ($64 \rightarrow 64$) $\rightarrow$ SiLU $\rightarrow$ Outputs).
- **Message embedding function** ($\mathbf{MLP}_{\mathrm{trans}}$): Three linear layers with SiLU activation (Inputs $\rightarrow$ Linear ($64 \rightarrow 64$) $\rightarrow$ SiLU $\rightarrow$ Linear ($64 \rightarrow 64$) $\rightarrow$ Linear ($64 \rightarrow 1$) $\rightarrow$ Outputs).

In all experiments, we utilize 2-layer classifiers.

Training Details. Our method is implemented on Python 3.10, PyTorch 2.0.1, and Torch-geometric 2.4.0. All experiments are conducted using four 24GB NVIDIA GeForce RTX 4090 GPUs. For all experiments, we set the number of rounds (R) to 100 and the number of local epochs to 1. This setting is applied consistently across all baselines.

Hyperparameters. We set the number of learnable nodes s to 20, the tail-degree threshold γ to 3, and select the regularization parameter β to values in the range of [0.01, 0.1, 1].

M DETAILED PROCESS OF GENERATING HH/HT/TH/TT GLOBAL SYNTHETIC DATA

In this section, we describe the process of generating global synthetic data using **1)** Head Class & Head Degree nodes (HH), **2)** Head Class & Tail Degree nodes (HT), **3)** Tail Class & Head Degree nodes (TH), and **4)** Tail Class & Tail Degree nodes (TT). FedLoG has two branches, each generating $\mathcal{V}_{k,\mathrm{head}}$ and $\mathcal{V}_{k,\mathrm{tail}}$, which contain knowledge from head degree nodes and tail degree nodes, respectively.

HH.. As described in Section 3.2, we generate HH global synthetic data by merging the head degree condensed nodes $\mathcal{V}_{k,\text{head}}$ from all clients, weighted by the proportion of head classes for each client. In Figure 1(d), for each class $c \in C$, the feature vector of the i-th global synthetic node for class c, $x_{v_g^{(c,i)}}$, is defined as:

$$x_{v_g^{(c,i)}} = \frac{1}{\sum_{k=1}^{K} r_k^c} \sum_{k=1}^{K} r_k^c x_{v_{k,\text{head}}^{(c,i)}},$$

where $r_k^c = \frac{|\mathcal{V}_k^c|}{|\mathcal{V}_k|}$ represents the proportion of nodes labeled c in the k-th client's dataset.

HT.. In generating HT global synthetic data, we substitute $\mathcal{V}_{k,\text{head}}$ with $\mathcal{V}_{k,\text{tail}}$. Thus, for each class $c \in C$, the feature vector of the i-th global synthetic node for class c, $x_{v_g^{(c,i)}}$, is defined as:

$$x_{v_g^{(c,i)}} = \frac{1}{\sum_{k=1}^{K} r_k^c} \sum_{k=1}^{K} r_k^c x_{v_{k,\text{tail}}^{(c,i)}},$$

TH.. For generating TH global synthetic data, we aim to give more weight to the tail classes. To achieve this, we adjust the weights inversely proportional to r_k^c, ensuring that tail classes (with lower r_k^c) receive higher weights. The new equation is given by:

$$x_{v_g^{(c,i)}} = \frac{1}{\sum_{k=1}^{K} \alpha_k^c} \sum_{k=1}^{K} \alpha_k^c x_{v_{k,\text{head}}^{(c,i)}}, \text{ where } \alpha_k^c = \frac{\sum_{j=1}^{K} r_j^c}{r_k^c + \epsilon} \tag{19}$$

Here, ϵ is a very small positive value added to prevent division by zero. In this revised equation, α_k^c assigns higher weights to classes with smaller r_k^c values, thereby giving more importance to the tail classes.

TT.. Finally, we generate TT global synthetic data using:

$$x_{v_g^{(c,i)}} = \frac{1}{\sum_{k=1}^{K} \alpha_k^c} \sum_{k=1}^{K} \alpha_k^c x_{v_{k,\text{tail}}^{(c,i)}}, \text{ where } \alpha_k^c = \frac{\sum_{j=1}^{K} r_j^c}{r_k^c + \epsilon} \tag{20}$$

N NOTATIONS

In this section, we summarize the main notations used in this paper. Table 8 provides the main notations and their descriptions. For simplicity, we describe the notation based on a head-branch.

Table 8: Summary of the notations

Notation	Description
General Notations	
S	Server
K	Number of clients
$\mathcal{G}$	Graph
$\mathcal{V}$	Set of nodes
$\mathcal{E}$	Set of edges
$\mathcal{D}$	Dataset consisting of $\mathcal{G}$ and Y
Y	Label set for the nodes
$X_{\mathcal{V}}$	Feature matrix of a set of nodes $\mathcal{V}$
x_v	Feature vector of a node $v \in \mathcal{V}$
ϕ	Set of parameters of a global model
$\lvert C_{\mathcal{V}} \rvert$	Number of classes within a set of nodes $\mathcal{V}$
r	Current round
h_v	Representation of a node v
α	Weight of prediction between head and tail branches
Local Client k Notations	
$\mathcal{G}_k$	Local graph for client k
$\mathcal{V}_k$	Set of nodes within a local graph for client k
$\mathcal{E}_k$	Set of edges within a local graph for client k
$\mathcal{D}_k$	Local dataset for client k
$\mathcal{D}_{\text{local}}$	Combined local datasets, $\mathcal{D}_{\text{local}} = \bigcup_{k=1}^{K} \mathcal{D}_k$
Y_k	Label set for the nodes within a local graph for client k
ϕ_k	Set of parameters of a local model for client k
$\varphi_{k,E}$	Parameters of a GNN embedder for client k
$\varphi_{k,H}$	Parameters of a head-branch classifier for client k
$\varphi_{k,T}$	Parameters of a tail-branch classifier for client k
$\mathcal{V}_{k,\text{head}}$	Set of synthetic nodes within a head-degree branch of client k
$v_{k,\text{head}}^{(c,s)}$	s-th synthetic node for class c in a head-branch for client k
$X_{\mathcal{V}_{k,\text{head}}}$	Feature matrix of a set of nodes $\mathcal{V}_{k,\text{head}}$
$x_{k,head}^{(c,s)}$	Feature of the s-th synthetic node for class c in a head-branch for client k
$h_{k,head}^{(c,s)}$	Representation of the s-th synthetic node for class c in a head-branch for client k
$P_{k,head}$	Set of prototype representations (i.e., representations of synthetic nodes) in a head-branch for client k
$\bar{h}_{\mathcal{V}_{k,\text{head}}^c}$	Average of prototype representations of class c in a head-branch for client k
r_k^c	Proportion of nodes labeled c in client k's dataset $\mathcal{D}_k$
$\mathbf{NG}_k$	Pretrained neighbor generator for client k (regardless of specific class)
γ_k	Class-wise adaptive factor for client k
$\mathcal{G}_k^{\text{syn}}$	Synthetic graph set consisting of graphs, each containing the global synthetic nodes (which are adapted locally) neighboring with their generated neighbor nodes.
$\mathcal{N}_{v_k}$	Generated neighbor node of node v_k
$\hat{x}_{\mathcal{N}_{v_k}}$	Generated feature of generated neighbor node $\mathcal{N}_{v_k}$ for the input feature x_{v_i} of node v_k
$x_{\mathcal{N}_{v_k}}$	Average of features of the h-hop neighbors of node v_k within $\mathcal{G}_k$
Global Synthetic Node Notations	
$\mathcal{D}_g$	Global synthetic dataset
$\mathcal{G}_g$	Global synthetic graph
$\mathcal{V}_g$	Set of global synthetic nodes
Y_g	Label set for the global synthetic nodes
$X_{\mathcal{V}_g}$	Feature matrix of the global synthetic nodes $\mathcal{V}_g$
$x_{v_g^{(c,s)}}$	Feature of the s-th global synthetic node v_g for class c
$\mathbf{NG}^c$	Class-specific neighbor generator for class c (by aggregating $\mathbf{NG}_k$ for all k in a class-wise manner)
Hyperparameter Notations	
s	Hyperparameter for assigning the number of synthetic nodes per class
λ	Hyperparameter for adjusting tail degree threshold
β	Hyperparameter for adjusting the extent of regularization of the features of synthetic nodes

O EXPERIMENTAL DATASET STATISTICS

In this section, we provide the experimental dataset statistics for all testing settings for three clients, allowing for an easy verification of the data distribution of each client and the New Client. In the 'Global' row, we sum up the statistics from all local clients.

Table 9: Cora Dataset Statistics (Closed Set)

		Seen Graph			Unseen Node	Missing Class
Dataset	**Class**	**Train**	**Valid**	**Test**	**Test**	**Test**
Global	0	49	37	32	268	86
	1	82	45	75	258	0
	2	140	110	133	217	220
	3	242	206	171	605	0
	4	120	82	90	268	0
	5	45	41	32	120	29
	6	53	42	27	9	59
Client 0	0	8	12	4	121	0
	1	7	4	3	124	0
	2	4	2	3	190	0
	3	208	168	131	125	0
	4	33	22	22	96	0
	5	0	0	0	0	29
	6	0	1	0	1	23
Client 1	0	0	0	0	0	86
	1	7	1	3	117	0
	2	136	108	130	27	0
	3	6	15	14	202	0
	4	74	49	60	74	0
	5	3	4	2	44	0
	6	0	0	0	0	36
Client 2	0	41	25	28	147	0
	1	68	40	69	17	0
	2	0	0	0	0	220
	3	28	23	26	278	0
	4	13	11	8	98	0
	5	42	37	30	76	0
	6	53	41	27	8	0
New Client	0	-	-	222	-	-
	1	-	-	12	-	-
	2	-	-	2	-	-
	3	-	-	107	-	-
	4	-	-	87	-	-
	5	-	-	166	-	-
	6	-	-	9	-	-

Table 10: CiteSeer Dataset Statistics (Closed Set)

		Seen Graph			Unseen Node	Missing Class
Dataset	**Class**	**Train**	**Valid**	**Test**	**Test**	**Test**
Global	0	41	24	25	72	0
	1	90	75	90	185	0
	2	196	163	137	424	93
	3	116	85	77	210	0
	4	154	96	110	132	121
	5	37	25	26	88	53
Client 0	0	19	5	11	26	0
	1	52	50	58	59	0
	2	67	52	32	296	0
	3	73	52	44	76	0
	4	7	1	7	60	0
	5	0	0	0	0	53
Client 1	0	5	5	2	18	0
	1	27	16	26	79	0
	2	129	111	105	128	0
	3	16	15	14	66	0
	4	0	0	0	0	121
	5	25	13	17	44	0
Client 2	0	17	14	12	28	0
	1	11	9	6	47	0
	2	0	0	0	0	93
	3	27	18	19	68	0
	4	147	95	103	72	0
	5	12	12	9	44	0
New Client	0	-	-	35	-	-
	1	-	-	53	-	-
	2	-	-	12	-	-
	3	-	-	110	-	-
	4	-	-	93	-	-
	5	-	-	211	-	-

Table 11: PubMed Dataset Statistics (Closed Set)

		Seen Graph			Unseen Node	Missing Class
Dataset	**Class**	**Train**	**Valid**	**Test**	**Test**	**Test**
Global	0	1271	983	949	221	168
	1	1176	897	951	572	1003
	2	2972	2171	2141	658	0
Client 0	0	277	222	209	93	0
	1	0	0	0	0	346
	2	1642	1227	1189	193	0
Client 1	0	994	761	740	128	0
	1	0	0	0	0	657
	2	594	408	414	213	0
Client 2	0	0	0	0	0	168
	1	1176	897	951	572	0
	2	736	536	538	252	0
New Client	0	-	-	787	-	-
	1	-	-	3500	-	-
	2	-	-	591	-	-

Table 12: Photos Dataset Statistics (Closed Set)

		Seen Graph			Unseen Node	Missing Class
Dataset	**Class**	**Train**	**Valid**	**Test**	**Test**	**Test**
Global	0	150	102	108	61	28
	1	592	502	468	881	0
	2	266	219	197	61	20
	3	358	241	258	329	0
	4	300	258	250	327	0
	5	332	217	247	0	25
	6	214	146	128	907	0
	7	39	17	27	307	0
Client 0	0	0	0	0	0	28
	1	71	50	49	345	0
	2	261	215	196	3	0
	3	49	26	33	135	0
	4	5	6	10	186	0
	5	332	217	247	0	0
	6	9	5	7	96	0
	7	9	5	9	59	0
Client 1	0	146	100	107	7	0
	1	394	369	333	226	0
	2	0	0	0	0	20
	3	11	8	8	101	0
	4	3	1	0	67	0
	5	0	0	0	0	18
	6	197	132	113	69	0
	7	1	0	3	40	0
Client 2	0	4	2	1	54	0
	1	127	83	86	310	0
	2	5	4	1	58	0
	3	298	207	217	93	0
	4	292	251	240	74	0
	5	0	0	0	0	7
	6	8	9	8	742	0
	7	29	12	15	208	0
New Client	0	-	-	0	-	-
	1	-	-	72	-	-
	2	-	-	3	-	-
	3	-	-	43	-	-
	4	-	-	64	-	-
	5	-	-	1	-	-
	6	-	-	1412	-	-
	7	-	-	248	-	-

Table 13: Computers Dataset Statistics (Closed Set)

		Seen Graph			Unseen Node	Missing Class
Dataset	**Class**	**Train**	**Valid**	**Test**	**Test**	**Test**
Global	0	168	120	115	146	140
	1	297	221	201	1604	0
	2	558	442	410	2	479
	3	91	63	66	765	0
	4	1399	1094	1103	3397	0
	5	129	69	98	0	60
	6	182	134	164	132	93
	7	343	220	232	4	167
	8	748	597	580	1321	0
	9	119	80	79	105	22
Client 0	0	164	118	114	32	0
	1	98	65	71	398	0
	2	558	442	410	2	0
	3	66	42	46	206	0
	4	41	25	30	813	0
	5	0	0	0	0	46
	6	0	0	0	0	93
	7	343	220	232	4	0
	8	12	10	8	447	0
	9	108	59	61	25	0
Client 1	0	4	2	1	114	0
	1	125	84	80	573	0
	2	0	0	0	0	262
	3	7	6	5	235	0
	4	139	123	117	1504	0
	5	129	69	98	0	0
	6	181	133	164	4	0
	7	0	0	0	0	143
	8	707	561	552	178	0
	9	11	21	18	80	0
Client 2	0	0	0	0	0	140
	1	74	72	50	633	0
	2	0	0	0	0	217
	3	18	15	15	324	0
	4	1219	946	956	1080	0
	5	0	0	0	0	14
	6	1	1	0	128	0
	7	0	0	0	0	24
	8	29	26	20	696	0
	9	0	0	0	0	22
New Client	0	-	-	30	-	-
	1	-	-	1374	-	-
	2	-	-	2	-	-
	3	-	-	302	-	-
	4	-	-	1360	-	-
	5	-	-	1	-	-
	6	-	-	1	-	-
	7	-	-	1	-	-
	8	-	-	167	-	-
	9	-	-	5	-	-