ORIGINAL ARTICLE
Journal Section

Measuring and Sampling: A Metric-guided Subgraph Learning Framework for Graph Neural Network

Jiyang Bai[1*] | Yuxiang Ren[2*] | Jiawei Zhang[3]

[1]Department of Computer Science, Florida State University, Tallahassee, Florida, 32306, USA. Email: bai@cs.fsu.edu

[2]IFM Lab, Department of Computer Science, Florida State University, Tallahassee, Florida, 32306, USA. Email: yuxiang@ifmlab.org

[3]IFM Lab, Department of Computer Science, University of California, Davis, Davis, California, 95616, USA. Email: jiawei@ifmlab.org

Correspondence
Yuxiang Ren, IFM Lab, Department of Computer Science, Florida State University, Tallahassee, Florida, 32306, USA
Email: yuxiang@ifmlab.org

Funding information
National Science Foundation, Grant Number: IIS-1763365

Graph neural networks (GNNs) have shown convincing performance in learning powerful node representations that preserve both node attributes and graph structural information. However, many GNNs encounter problems in effectiveness and efficiency when they are designed with a deeper network structure or handle large-sized graphs. Several sampling algorithms have been proposed for improving and accelerating the training of GNNs, yet they ignore understanding the source of GNNs performance gain. The measurement of information within graph data can help the sampling algorithms to keep high-value information while removing redundant information and even noise. In this paper, we propose a **Me**tric-**Guide**d (MeGuide) subgraph learning framework for GNNs. MeGuide employs two novel metrics: *Feature Smoothness* and *Connection Failure Distance* to guide the subgraph sampling and mini-batch based training. *Feature Smoothness* is designed for analyzing the feature of nodes in order to retain the most valuable information, while *Connection Failure Distance* can measure the structural information to control the size of subgraphs. We demonstrate the effectiveness and efficiency of MeGuide

*Should be considered joint first author

in training various GNNs on multiple datasets.

KEYWORDS
Graph Neural Network; Subgraph Sampling; Training Optimization;

1 | INTRODUCTION

In recent years, graph neural networks (GNNs) have progressed in graph representation learning. Numerous real-world graph-related applications, such as social media [1], fake news detection [2] and knowledge graphs [3], exhibit the favorable property of graph neural networks. The core idea of GNN is to aggregate the feature information of the nodes' neighbors through neural networks to update node representations, which combine both the independent information of the nodes and corresponding structural information. As the scale of the graph increases and the GNN model architecture goes deeper, several challenging problems will emerge in learning GNNs: *neighbors explosion*, *node dependence*, and *oversmoothing* [4]. GNNs learn high-level representations through a recursive neighbors aggregation scheme [5], which causes the number of neighbors to explode. This problem is described as *neighbors explosion* [4], which leads to exponentially-growing computation complexity. Because neighboring nodes are interdependent in the learning process, most current GNNs have to work based on the full graph. This limitation is the *node dependence* as described in [6], which brings out serious memory and computation bottlenecks in handling large-sized graphs. At last, when GNNs go deeper and learn on the full graph, node representations from different clusters will be aggregated [7], which contradicts the smoothness assumptions (close nodes are similar). This *oversmoothing* issue can lead to learned node representations indistinguishable [8, 9].

Several sampling-based methods have been proposed to deal with the aforementioned problems. Among them, one direction is the node-sampling method [10, 11, 12, 13, 9]. GraphSAGE [10] samples features from local node neighborhoods to learn a function that updates node representations. The neighborhoods sampling can mitigate the harassment from *neighbors explosion* and *node dependence*. DropEdge [9] randomly removes some edges from the input graphs, which is a kind of node-sampling essentially, to overcome the *oversmoothing*. However, for large-sized graphs, even if neighboring nodes are sampled, it is still difficult to avoid loading the full graph during training, which brings huge memory overhead. At the same time, the node-sampling methods focus on sampling distribution (e.g., importance sampling [13]) to reduce the high-level approximation variance, but such sampling cannot precisely quantify the information gain from each node.

The other direction is the graph-sampling method [6, 14, 4], which can deal with the memory overhead from loading the full graph. Cluster-GCN [6] constructs the subgraph mini-batches by clustering on the full graph to train the GCN. However, the clustering algorithm itself is greatly affected by the structure of the full graph, that is, the size of the clustered subgraph is uncontrollable. As a heuristic method, Cluster-GCN is difficult to guarantee the generalization performance of graph data with different structures. Unlike Cluster-GCN, GraphSAINT [14] constructs mini-batches by sampling subgraphs with the support of several random samplers. RippleWalk [4] proposes a training framework together with the ripple walk sampler to consider randomness and connectivity of sampled subgraphs. Graph-sampling methods using subgraphs to train GNNs can handle the three problems mentioned above simultaneously to a certain extent but only focus on the training phase of GNN models. GraphSAINT and RippleWalk still need to load the full graph when using the trained model to make predictions, which challenges the memory space. What is more, current graph-sampling methods along with their samplers are limited in understanding the source of information gains and noise when GNNs are trained. In fact, the understanding of information gain and noise can effectively help graph-sampling methods to obtain higher quality subgraphs that are the key to the training performance.

In this paper, we propose a general learning framework, namely **Me**tric-**Guide**d (MeGuide), for graph neural networks. MeGuide is a mini-batch based learning framework that can help GNNs learn on sampled subgraphs instead of the full graph. The full GNN is trained and updated based on the mini-batch gradient. MeGuide employs a novel MeGuide Sampler to sample subgraphs for the mini-batch, whose sampling logic is mainly based on the two novel metrics we introduce in this paper: *Feature Smoothness* and *Connection Failure Distance*. In subgraph sampling, the node selection and the size of the subgraph are two important factors that determine the shape of the subgraph. For node selection, the existing methods [9, 14] randomly sample neighbors and ignore the information provided by node feature vectors. *Feature Smoothness* measures the information gain from node features, which is used by MeGuide Sampler to select nodes with more information and drop nodes with redundant information during the subgraph sampling process. In addition, according to the basic assumption on the graph structure data, the closer the nodes are, the more similar and the more likely they are to have the same label. It is reasonable to consider that neighbors with different labels contribute to negative disturbance [15]. When the distance between nodes exceeds a certain metric, there is a greater probability for the sampled nodes to have different labels. This metric we define in this paper is *Connection Failure Distance*, which is utilized by MeGuide Sampler to control the longest multi-hop connection in subgraphs. In this way, MeGuide can help GNNs avoid unexpected aggregations (e.g., aggregating nodes with different labels), which is the primary cause of *oversmoothing* [4]. To a certain extent, the size of subgraphs can also be determined, but in other sampling methods, it is a hyper-parameter. With the understanding of information gains and disturbance during sampling subgraphs, MeGuide can help the training of GNN models to achieve better performance in both effectiveness and efficiency. In addition, for how to apply trained GNN models to perform prediction tasks based on subgraphs, MeGuide proposes a representation aggregation-based scheme so that it is no longer necessary to load the full graph in the prediction phase.

The contributions of our work are summarized as follows:

- We define two metrics *Feature Smoothness* and *Connection Failure Distance* based on the smoothness of node features and the connectivity of graphs respectively to measure the quantity and quality of information gain between nodes.
- We propose a general learning framework MeGuide for different GNN models. MeGuide samples high-value subgraphs guided by *Feature Smoothness* and *Connection Failure Distance* to train GNN models.
- For the case of the memory bottleneck when using learned GNN models to make predictions on a single large graph, MeGuide employs the representation aggregation-based prediction on subgraphs, which is an unconsidered problem left by existing graph-sampling training methods.
- We conduct extensive experiments on 5 benchmark graph datasets with different sizes to demonstrate both the effectiveness and efficiency of MeGuide. The results show the superiority of MeGuide subject to the training efficiency and the performance of prediction.

The remaining paper is organized as follows. We review the related works in Section 2. Then we introduce preliminaries and background in Section 3. The proposed framework MeGuide is introduced in Section 4, whose effectiveness is evaluated in Section 5. Finally, we conclude this paper in Section 6.

2 | RELATED WORK

2.1 | Graph Neural Network

Graph neural networks (GNNs) aim at the machine learning tasks involving graph-structured data. Bruna et al. [16] express the idea of graph information construction based on the theorem of the spectrum of the graph Laplacian. Later, spatial-based GNNs [17, 18] define graph convolutions directly based on a node's spatial relations. Different from the spectral-based GNNs, where the weights of edges in the graph have been determined before the training, the connections (edges) between nodes and the weights of connections can be automatically learned in the training process. For example, GAT [17] applies the attention mechanism to learn the attention weights for edges in each training epoch. Nonetheless, both the spectral-based and spatial-based GNNs can be regarded as an information propagation-aggregation mechanism, and such mechanism is achieved by the connections and multi-layers structure. As a popular research topic, there have been many GNNs [19, 10, 20, 21, 22, 23] showing awe-inspiring capabilities in handling graph structure data. More progress on graph neural networks can be referred to the surveys [24, 25].

2.2 | Optimization in GNN

Many GNNs are limited by the problems from three aspects: *node dependence*, *neighbors explosion*, and *oversmoothing*. In response to these problems, some related works have been proposed from different directions.

2.2.1 | Node Dependence

Node dependence [6] forces GNNs to be trained on the full graph, which leads to slow training process. To deal with such problem, The works [6, 14] apply the concept of subgraph training methods. The essence of subgraph training is to collect a batch of subgraphs from the full graph and use them during the training process. The subgraph collecting strategies can be various within different methods. Chiang et al. [6] divide the full graph into subgraphs according to the clustering results. Training GNNs by clustered subgraphs can avoid unexpected aggregations from different clusters, but the size of clustered subgraphs is uncontrollable. More importantly, the subgraphs sampled by clustering share no joint node, thus the connection information within the full graph will be partly discarded. Another work in [14] applies several subgraph sampling ideas (e.g., on the node, edge, random walk) when training the GCN for inductive tasks. There are also other approaches to optimize the GNNs frameworks. [26, 19, 27, 28] optimize the localized filter in order to reduce the time cost of training on the full graph. Further, [29, 30] reduce the number of learnable parameters by dimensionality reduction and residual graph laplacian respectively. But these approaches do not alleviate the space complexity problem.

2.2.2 | Neighbors Explosion

Neighbors explosion makes deep GNNs difficult to being implemented. Because learning a single node requires embeddings from its neighbors, and the quantity may be explosive when a GNN goes deeper. Some research works deal with *neighbors explosion* by neighbors sampling [10, 11, 31, 12]. GraphSAGE [10] proposes to sample neighbors when aggregating information for every node. FastGCN [11] regards the neighbors following specific distribution, and then does neighbor sampling on the distribution level. VR-GCN [12] conducts the neighbors sampling with variance reduction for each node. In other directions, several models [32, 33, 34] select specific neighbors based on defined metrics to avoid the explosive quantity. In [35, 36, 37, 38], the propagation-aggregation mechanism is optimized to enable the node to capture long-distance information even with a relatively shallow structure. DropEdge [9] randomly remove edges from input graphs to handle the *neighbor explosion*. The above related works all focus on the neighbor-level sampling but still have the same space complexity with original GNNs.

2.2.3 | Oversmoothing

The problem of *oversmoothing* in GNNs was introduced in [8]. When GNNs go deep, the performance suffers from *oversmoothing* where node representations from different clusters become mixed up [7]. The node information propagation-aggregation mechanism can be regarded as one type of random walk within the graph. With the increasing of walking steps, the node representations will finally converge to a stable status. Such convergence would impede the performance of GNNs and make the nodes indistinguishable in the downstream tasks. Some related works have been proposed to deal with the *oversmmothing*. GResNet [39] comes up with the suspended animation and utilizes the residual networks to mine the advantages of deeper networks. In such a case, the depth of GNN models can reach more than fifty with a better performance.

Till now, the optimized methods based on subgraph learning are limited [14, 6, 4]. Their measurement of the quality of sampled subgraphs is not comprehensive enough, and the sampling of subgraphs is precisely the key to this type of method. In this paper, we propose a subgraph learning framework MeGuide for GNNs, which employs two novel metrics: *Feature Smoothness* and *Connection Failure Distance* to guide the subgraph sampling and mini-batch based training. High-quality subgraphs can ultimately help MeGuide to better empower different GNNs learning.

3 | PRELIMINARIES AND BACKGROUND

In this section, we first introduce the preliminaries about general GNN models. Then we elaborate the basic idea of subgraph-based training for GNN models.

3.1 | General GNN Models

We denote a graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ represent the set of nodes and edges of $\mathcal{G}$ respectively. Most widely used GNN mdoels (e,g, GAT [17], GCN [19], GIN [5]) follow the recursive neighbors aggregation scheme to update the representation for each node. For node v_i, we use $\mathcal{N}_{v_i} = \{v_j : e_{v_i,v_j} \in \mathcal{E}\}$ to represent the set of its neighbors, where e_{v_i,v_j} denotes the edge between v_i and v_j. Each node has an initial feature vector $x_v \in \mathbb{R}^d$ with dimension d. The initial feature matrix $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$ is consitute of all nodes' initial feature vectors. The neighbors aggregation scheme of GNN models can be represented generally as:

$$\begin{aligned} &\mathbf{H}^{(0)} = \mathbf{X} \\ &\mathbf{H}_{v_i}^{(l+1)} = \sigma\Big(\sum_{j \in \mathcal{N}_{v_i}} \alpha_{ij} \cdot \mathbf{H}_{v_j}^{(l)} \mathbf{W}^{(l)}\Big) \end{aligned} \tag{1}$$

Here, $\mathbf{H}^{(0)}$ is the input feature matrix of a GNN model, and the $\mathbf{H}_{v_j}^{(l)}$ is the hidden representation of node v_j in the l_{th} layer; σ is an activation function; $\mathbf{W}^{(l)}$ is the learnable parameter matrix for linear transformation; α is a variant of cofficient matrix, which has different definitions according to different GNN models. For example, in GCN [19], $\alpha = \widetilde{\mathbf{A}}$ is the normalized adjacency matrix. When in GAT [17], α is the attention weights matrix learned in current round.

Through the feedforward layer computing, the hidden representation of node v_i is updated by aggregating its current representation and neighbors' hidden representations. With the support of a mapping function (e.g. a fully-connected layer), the learned representation can serve for downstream tasks such as node classification.

3.2 | Subgraph-based training for GNN models

The learning scheme shown in Equation 1 needs to take the full graph $\mathcal{G}$ as input. Training GNN models with the full graph, especially facing large-sized graphs, may easily lead to aforementioned three problems: *neighbors explosion*, *node dependence*, and *oversmoothing* [4]. To solve these problems, subgraph-based training methods [4, 14] employ subgraphs of $\mathcal{G}$ to construct the mini-batch in each training iteration and update the complete GNN models based on the mini-batch gradient. We use the $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$ to denote a subgraph of $\mathcal{G}$, where $\mathcal{V}_t \subseteq \mathcal{V}$ and $\mathcal{E}_t \subseteq \mathcal{E}$; In this way, the neighbor aggregation procedure of GNN models when training with the subgraph $\mathcal{G}_t$ can be represented as:

$$\begin{aligned} \mathbf{H}^{(0)} &= \mathbf{X}_{\mathcal{G}_t} \\ \mathbf{H}_{v_i}^{(l+1)} &= \sigma\Big(\sum_{j \in \mathcal{N}_{v_i}^{\mathcal{G}_t}} \alpha_{ij}^{\mathcal{G}_t} \cdot \mathbf{H}_{v_j}^{(l)} \mathbf{W}^{(l)} \Big) \end{aligned} \tag{2}$$

Here, $\mathcal{N}_{v_i}^{\mathcal{G}_t}$ is the neighbor nodes set of node v_i in $\mathcal{G}_t$; $\alpha^{\mathcal{G}_t}$ corresponds to the cofficient matrix of $\mathcal{G}_t$. The aggregated representations of nodes in $\mathcal{G}_t$ are used to calculate the loss and gradients in order to update the complete GNN models.

However, different from previous data types such as the image using mini-batch gradient descent, the nodes in a graph are not independent from each other. In this way, sampling subgraphs is equivalent to dropping some edges, which may lead to losing dependency information (connections). For this concern, RippleWalk [4] provides a theoretical analysis in Theorem 1 and 2, which prove that the subgraph-based mini-batch gradient descent is still reliable for optimizing GNN models. However, the quality of subgraphs will have a great impact on the training performance, and this is also the problem we try to solve in the paper: sampling more effective subgraphs for the learning of GNN models.

4 | PROPOSED METHODOLOGY

In this section, we first introduce two metrics *Feature Smoothness* and *Connection Failure Distance* that are the keys to sampling effective subgraphs. Then we propose the metric-guided sampling method, and follow it up by presenting the subgraph-based training and representation aggregation-based prediction of the proposed MeGuide framework.

4.1 | Subgraph Sampling Metrics

The neighbors aggregation scheme works to collect and aggregate neighboring information to update node representations. The neighboring relationship on a graph can indicate the closeness among nodes to a certain extent. We analyze the neighboring information aggregation process from the perspective of information gain [40]. When neighboring nodes $\mathcal{N}_{v_i}$ and v_i are too similar, they cannot bring much information gain to the final aggregated representation of v_i. At the same time, the neighbors aggregation scheme will aggregate the multi-hop neighboring nodes with more stacked convolution layers. When the hop distance is too long, node representations from different clusters become mixed up [7], which is the primary reason of *oversmoothing*. Therefore, the hop distance should be related to the effectiveness of information gain during the neighbors aggregation process.

Based on the above two analyses, we define *Feature Smoothness* and *Connection Failure Distance* to measure the quantity and quality of information gain in the aggregation process, respectively. These two metrics can guide the sampling method to obtain subgraphs that can bring high-quality information gain for the learning of GNN models.

4.1.1 | Feature Smoothness

In order to quantify the information obtained from neighboring nodes, we use Kullback-Leibler divergence to measure the information gain between two connected nodes.

Definition 1 *(Information Gain between Connected Nodes): Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, for each node $v \in \mathcal{V}$ and its representation follows the distribution Q; the node $v' \in \mathcal{N}_v$, and the aggregated representation of v from v' (denoted as $AGG(v, v')$), follows the distribution Q_{AGG}. Assume Q and Q_{AGG} are over the same feature space $\mathcal{X}$. The information gain of the graph $\mathcal{G}$ can be measured by the Kullback-Leibler divergence [40] as:*

$$D_{KL}(Q_{AGG}||Q) = \int_{\mathcal{X}} Q_{AGG}(x) \cdot \log \frac{Q_{AGG}(x)}{Q(x)} dx \tag{3}$$

Similarly, for a specific node $v_i \in \mathcal{V}$ and its neighboring node $v_j \in \mathcal{N}_{v_i}$, we can denote the information gain of v_i from v_j as $D_{KL}(Q_{AGG(v_i,v_j)}||Q_{v_i})$.

From Hou et al. [15], the Kullback-Leibler divergence can measure the information gain between one node and all neighboring nodes. Since for node v, if the features of its neighboring nodes are exactly the same as v's feature, obviously the divergence is 0. On the other hand, when the neighboring nodes possess significantly different feature distributions compared with the central node, the divergence is relatively large. However, in practice, the accurate distributions of node features are unknown. Therefore we propose the following *Feature Smoothness* to quantify the actual information gain.

Definition 2 *(Feature Smoothness): Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the feature smoothness of the graph is defined as:*

$$\lambda_f = \frac{|| \sum_{v \in \mathcal{V}} \left(\sum_{v' \in \mathcal{N}_v} (x_v - x_{v'}) \right)^2 ||_1}{|\mathcal{E}| \cdot d} \tag{4}$$

where $|| \cdot ||_1$ is the Manhattan norm, $x_v \in \mathbb{R}^d$ and $x_{v'} \in \mathbb{R}^d$ are the initial features of v and v', respectively. Similarly, we can also define the feature smoothness between connected nodes v_i and v_j as:

$$\lambda_{f^{(v_i,v_j)}} = \frac{||(x_{v_i} - x_{v_j})^2||_1}{d} \tag{5}$$

The λ_f in Definition 2 counts the sum of norm-2 distance among connected nodes, which can provide an overall feature divergence of the entire graph. A higher λ_f indicates that the feature signals of a graph have *higher frequency* [15], meaning that the connected nodes in the graph are more likely dissimilar. While λ_f is an overall metric of the entire graph, the $\lambda_{f^{(v_i,v_j)}}$ measures the similarity between connected nodes v_i and v_j. Different from the Kullback-Leibler divergence that is unknown in practice, instead $\lambda_{f^{(v_i,v_j)}}$ can be easily calculated for specific connected nodes. Therefore, we propose to explicitly quantify the information gain with the help of $\lambda_{f^{(v_i,v_j)}}$, and state the relation between information gain and *Feature Smoothness* in the following theorem.

Theorem 1 *Given a $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the information gain of node v_i from its neighboring node v_j in Definition 1 is positively related to their feature smoothness $\lambda_{f(v_i,v_j)}$, i.e.,*

$$D_{KL}(Q_{AGG(v_i,v_j)}||Q_{v_i}) \sim \lambda_{f(v_i,v_j)} \tag{6}$$

Before proving Theorem 1, here we provide a lemma to assist the proof.

Lemma 1 *Assume Q is the distribution of node $v \in \mathcal{V}$ and S is the distribution of the neighboring nodes$\sum_{v' \in \mathcal{N}_v} v'$, the Kullback-Leibler divergence $D_{KL}(S||Q)$ is positively related to $Var(|\mathcal{N}_v| \cdot x_v - \sum_{v' \in \mathcal{N}_v} x_{v'})$, i.e.,*

$$D_{KL}(S||Q) \sim Var(|\mathcal{N}_v| \cdot x_v - \sum_{v' \in \mathcal{N}_v} x_{v'}) \tag{7}$$

where $Var(\cdot)$ denotes the variance, $|\mathcal{N}_v|$ is the number of nodes in $\mathcal{N}_v$.

Proof of Lemma 1 For $D_{KL}(S||Q)$, since the explicit formulas of S and Q are unknown, we use the discrete space approach: the histogram, to estimate S and Q. In detail, we divide the feature space $\mathcal{X} = [0,1]^d$ into r^d bins $\{H_1, H_2, \ldots H_{r^d}\}$ evenly, and the length is $\frac{1}{r}$ and dimension is d. Following the distributions of S and Q, there are $2|\mathcal{E}|$ corresponding samples of nodes (each connected two nodes can be regarded as the central node and the neighboring node, and vice versa) in total, we use the $|H_i|_Q$ and $|H_i|_S$ to denote the number of samples falling into bin H_i. In this way, we have

$$\begin{aligned} D_{KL}(S||Q) &\simeq \sum_{i=1}^{r^d} \frac{|H_i|_S}{2|\mathcal{E}|} \cdot \log \frac{\frac{|H_i|_S}{2|\mathcal{E}|}}{\frac{|H_i|_Q}{2|\mathcal{E}|}} \\ &= \frac{1}{2|\mathcal{E}|} \sum_{i=1}^{r^d} |H_i|_S \cdot \log \frac{|H_i|_S}{|H_i|_Q} \\ &= \frac{1}{2|\mathcal{E}|} (\sum_{i=1}^{r^d} |H_i|_S \cdot \log |H_i|_S - \sum_{i=1}^{r^d} |H_i|_S \cdot \log |H_i|_Q) \\ &= \frac{1}{2|\mathcal{E}|} (\sum_{i=1}^{r^d} |H_i|_S \cdot \log |H_i|_S - \sum_{i=1}^{r^d} |H_i|_S \cdot \log(|H_i|_S + \Delta_i)) \end{aligned} \tag{8}$$

where $\Delta_i = |H_i|_Q - |H_i|_S$. In this way, we can expand the term $\sum_{i=1}^{r^d} |H_i|_S \cdot \log(|H_i|_S + \Delta_i)$ by applying the second-order Taylor approximation at the point 0 as

$$\sum_{i=1}^{r^d} |H_i|_S \cdot \log(|H_i|_S + \Delta_i) \simeq \sum_{i=0}^{r^d} |H_i|_S (\log |H_i|_S + \frac{\ln 2}{|H_i|_S} \cdot \Delta_i - \frac{\ln 2}{2(|H_i|_S)^2} \cdot \Delta_i^2) \tag{9}$$

Note that the number of samples from S and Q are the same, which means

$$\sum_{i=0}^{r^d} |H_i|_S = \sum_{i=0}^{r^d} |H_i|_Q = 2|\mathcal{E}| \tag{10}$$

So we have $\sum_{i=0}^{r^d} \Delta_i = 0$. Therefore,

$$\begin{aligned} D_{KL}(S||Q) \simeq & \frac{1}{2|\mathcal{E}|}(\sum_{i=1}^{r^d} |H_i|_S \cdot \log |H_i|_S - \sum_{i=1}^{r^d} |H_i|_S \cdot \log(|H_i|_S + \Delta_i)) \\ \simeq & \frac{1}{2|\mathcal{E}|}(\sum_{i=1}^{r^d} |H_i|_S \cdot \log |H_i|_S - \sum_{i=0}^{r^d} |H_i|_S (\log |H_i|_S + \frac{\ln 2}{|H_i|_S} \cdot \Delta_i - \frac{\ln 2}{2(|H_i|_S)^2} \cdot \Delta_i^2)) \\ = & \frac{1}{2|\mathcal{E}|} \sum_{i=1}^{r^d} (\frac{\ln 2}{2|H_i|_S} \Delta_i^2 - \ln 2 \Delta_i) \\ = & \frac{\ln 2}{4|\mathcal{E}}| \sum_{i=1}^{r^d} \frac{\Delta_i^2}{|H_i|_S} \end{aligned} \tag{11}$$

For S and Q, we consider the samples of S as $\{x_v : v \in \mathcal{V}\}$ and samples of Q as $\{\frac{1}{|\mathcal{N}_v|} \sum_{v' \in \mathcal{N}_v} x_{v'} : v \in \mathcal{V}\}$ with counts $|\mathcal{N}_v|$ for node v. In this way, the difference between S and Q can be represented by the expectation of $|\mathcal{N}_v| x_v - \sum_{v' \in \mathcal{N}_v} x_{v'}$. Meanwhile, the discrepancy of numbers of samples in each bin, i.e. Δ_i, is positively correlated with the difference between S and Q. If we regard the $|H_i|_S$ as constant, we can infer

$$\begin{aligned} D_{KL}(S||Q) \simeq & \frac{\ln 2}{4|\mathcal{E}}| \sum_{i=1}^{r^d} \frac{\Delta_i^2}{|H_i|_S} \\ \simeq & \mathbb{E}[(|\mathcal{N}_v| x_v - \sum_{v' \in \mathcal{N}_v} x_{v'})^2] \\ = & Var(|\mathcal{N}_v| x_v - \sum_{v' \in \mathcal{N}_v} x_{v'}) \end{aligned} \tag{12}$$

Based on the Lemma 1, state the proof of Theorem 1.

Proof of Theorem 1 According to Lemma 1, for node v_i, S is the distribution of its neighboring nodes $\sum_{v' \in \mathcal{N}_{v_i}} x_{v'}$. Considering one neighboring node of v_i (e.g., $v_j \in \mathcal{N}_{v_i}$), the $D_{KL}(Q||S)$ in Lemma 1 will derive into $D_{KL}(Q_{v_i}||Q_{v_j})$, and it has

$$\begin{aligned} D_{KL}(Q_{v_j}||Q_{v_i}) \sim & Var(|\{v_j\}| \cdot x_{v_i} - \sum_{v' \in \{v_j\}} x_{v'}) \\ = & Var(x_{v_i} - x_{v_j}) \end{aligned} \tag{13}$$

According to the proof of Theorem 4 in [15] and our Definition 2,

$$\begin{aligned}\lambda_{f(v_i,v_j)} &= \frac{||(x_{v_i} - x_{v_j})^2||_1}{d} \\ &= \frac{||\sum_{v\in\{v_i\}}(\sum_{v'\in\{v_j\}} x_v - x_{v'})^2||_1}{|\{v_i\}|\cdot d} \\ &= \frac{||Var(x_{v_i} - x_{v_j})||_1}{d} \\ &\sim Var(x_{v_i} - x_{v_j})\end{aligned} \tag{14}$$

Therefore, we can find that

$$D_{KL}(Q_{v_j}||Q_{v_i}) \sim \lambda_{f(v_i,v_j)} \tag{15}$$

Finally, since the $AGG_{(v_i,v_j)}$ in Definition 1 is equivalent to weighted summation of x_{v_i} and x_{v_j} i.e., $x_{v_i} + \alpha_j x_{v_j}$, where $\alpha_j \in (0,1)$ is a fixed value, and $D_{KL}(Q_{v_i}||Q_{v_i}) = 0$, thus we can conclude

$$D_{KL}(Q_{AGG(v_i,v_j)}||Q_{v_i}) \sim D_{KL}(Q_{v_j}||Q_{v_i}) \sim \lambda_{f(v_i,v_j)} \tag{16}$$

From Theorem 1, we can conclude that a higher $\lambda_{f(v_i,v_j)}$ represents higher information between connected nodes. In such case, for a specific node v_i we are able to measure its information gain from the neighboring node v_j by computing the feature smoothness $\lambda_{f(v_i,v_j)}$. In Section 4.2, $\lambda_{f(v_i,v_j)}$ serves as the metric to determine the neighboring nodes sampling strategy, where the neighboring nodes can provide more information gain will be sampled.

4.1.2 | Connection Failure Distance

As we mentioned before, a long hop distance can lead to unexpected aggregation between nodes from different clusters. When measuring the aggregation of two nodes from different clusters by feature smoothness, the information gain may be relatively large, but the quality of this information gain is not high. Considering the node classification task, nodes from two different clusters are likely to have different labels. Aggregating the representations of nodes with different labels essentially introduces negative information to damage the discrimination of node representation, which results in *oversmoothing* finally. Therefore, in addition to measuring the quantity of information gain, we also need to explore the way to measure its quality.

Many previous works have analyzed through experiments that GNN models with deep stacking layers (long hop distances) bring very limited improvement in model performance [9, 41], and the model performance even declines significantly [42]. This actually shows that the longer the hop distance, the more negative information is introduced, and the quantity of negative information will eventually exceed the positive information. It is reasonable to consider that neighbors with the same label contribute positive information to the information gain and other neighbors contribute negative disturbance. As the hop distance between two nodes becomes longer, the probability of two nodes having different labels rises, and the possibility of introducing negative information is also rising. We treat multi-hop connections between nodes with different labels as failure connections because these connections bring more negative information gain. We provide a simple illustration of the failure connection in Figure 1. The different colors of the nodes in the graph represent different classes. The three connections marked in green are failed connections since

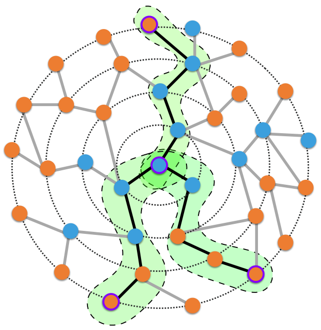

FIGURE 1 An illustration of the failure connection. The different colors of the nodes represent different node labels. The three connections marked in green are failed connections.

they link nodes with different labels. The longer the hop distance is, the probability that the central node encounters nodes with different labels increases. In this simple illustration, when the hop distance is greater than 3, the number of nodes with different labels will be far more than nodes with the same label. A large number of failure connections will make the negative information gain more than the positive one. Here we define a metric related to hop distance, namely *Connection Failure Distance*, to measure the quality of information gain.

Definition 3 *(Connnection Failure Distance): To measure the quality of the information gain brought by the neighbors aggregation process, we define the metric Connection Failure Distance as:*

$$\lambda_d = \frac{\sum_{v_i \in \mathcal{V}} \max\{hop\_dis(v_i, v_j) |\ v_j \in \mathcal{V}, \mathbb{I}(v_i, v_j) = 1\}}{|\mathcal{V}|} \tag{17}$$

where $\mathbb{I}(v_i, v_j)$ is an indicator function: if the label y_{v_i} of the node v_i is the same as y_{v_j} of the node v_j, it returns 1, otherwise returns 0. $hop\_dis(v_i, v_j)$ is the smallest number of hops between v_i and v_j. If two nodes can not be connected through multi-hops, we set the value as 0. λ_d reflects the average longest hop distance between two nodes with the same label in $\mathcal{G}$. Neighbors aggregation longer than λ_d is likely to be performed between two nodes with different labels, which contributes more negative information. Because of this, we name λ_d as *Connection Failure Distance* to indicate that the multi-hop connections longer than λ_d are meaningless.

Theorem 2 *Given two nodes v_i and v_j, if $hop\_dis(v_i, v_j) > \lambda_d$, $p(\mathbb{I}(v_i, v_j) = 1) < \frac{1}{\#classes}$. Here, $p(\cdot)$ represents the probability and $\#classes$ denotes the total number of classes of nodes.*

Proof According to the basic assumption in the graph structure data, the closer the nodes are, the more similar and the more likely they are to have the same label. From such assumption, given a v_i we can derive that $p(\mathbb{I}(v_i, v_j) = 1) > \frac{1}{\#classes}$ can always be satisfied when $hop\_dis(v_i, v_j)$ is small enough. It is obvious since nodes closer to v_i have the higher probability owning same label with v_i compared to other nodes, and such higher probability is greater than $\frac{1}{\#classes}$. Thus we can always find a d_0 that satisfies $d_0 < \lambda_d$ and $p(\mathbb{I}(v_i, v_j) = 1 | hop\_dis(v_i, v_j) <= d_0) > \frac{1}{\#classes}$. In such case, $p(\mathbb{I}(v_i, v_j) = 1 | hop\_dis(v_i, v_j) > d_0) < \frac{1}{\#classes}$ and then we can get $p(\mathbb{I}(v_i, v_j) = 1 | hop\_dis(v_i, v_j) > \lambda_d) < \frac{1}{\#classes}$.

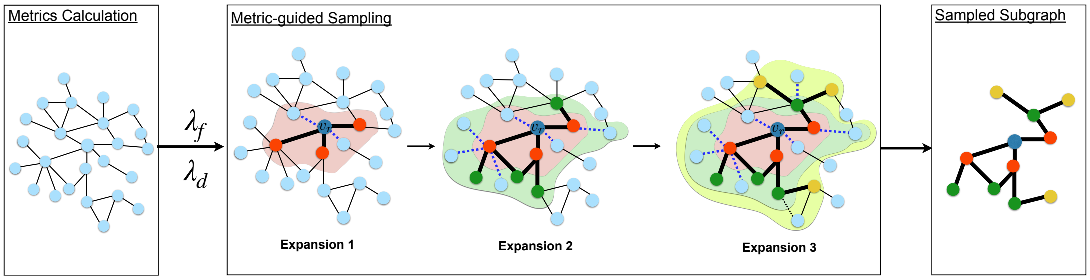

FIGURE 2 An illustration of the metric-guided sampling process of MeGuide sampler. In this example, $\lfloor \lambda_d/2 \rfloor = 3$ determines the expansion step is 3. The sampling process starts from one root node v_r. In each expansion, the background color region represents the neighbor set. For two connected nodes v_i and v_j, if $\lambda_{f^{(v_i,v_j)}} \geq \rho\lambda_f$, the edge between them is a bold black line and the node will be sampled. Otherwise, the edge is a blue dotted line, and the node will be dropped. After 3 expansion steps, the colored nodes represent the truly sampled nodes and constitute a subgraph. (Best viewed in color)

According to Theorem 2, given a specific node v_i, any node that has longer hop distance than λ_d is more likely to own different labels. Thus we can leverage λ_d as the metric to guide the subgraph sampling process, aiming to avoid the negative information aggregation within the sampled subgraphs.

However, if we use λ_d to measure the quality of the information gain, node labels are required for calculation. Since we cannot obtain the labels of all nodes for training, we utilize labeled nodes in the training set to estimate λ_d instead.

4.2 | Metric-Guided Subgraph Sampling

As a subgraph learning framework for GNN models, the subgraph sampling method is the key to the framework's performance. A suitable and accurate selection of neighboring nodes can retain node content information and graph topology at the same time, which can help achieve the same object as the full graph in Equation 2. Based on the proposed two metrics in Section 4.1, we design the metric-guided subgraph sampling method MeGuide Sampler to obtain effective subgraphs.

We utilize two previous proposed metrics as follows. At first, *Connection Failure Distance* λ_d is used to determine the overall size of subgraphs. MeGuide Sampler adopts a sampling strategy that expands from a random root node outwards sequentially, thus the number of expansion steps can control the scale of subgraphs to a certain extent. By setting the number of expansion steps to $\lfloor \lambda_d/2 \rfloor$, we can ensure that the hop distance between two nodes in the sampled subgraph does not constitute a failure connection, which can avoid much negative information. Here the value of λ_d is estimated by the labeled nodes in the training set.

Second, *Feature Smoothness* λ_f is used to select neighboring nodes with high-quantity information gain during the expansion process. As we prove in Section 4.1.1, the $\lambda_{f^{(v_i,v_j)}}$ can measure the information between two connected nodes. Here, we select the neighboring nodes with the condition $\lambda_{f^{(v_i,v_j)}} \geq \rho\lambda_f$, which means selecting those nodes with higher information gain than the overall level of the graph $\mathcal{G}$ but discard those with smaller information gain. Here, ρ is a hyper-parameter to control the feature smoothness-based selection criteria.

In Figure 2, we illustrate the sampling process with the *Connection Failure Distance* of 6, which enables a more intuitive understanding of each expansion. More details described by pseudocode are exhibited in Algorithm 1.

Algorithm 1: MeGuide Sampler

```
Input: Target graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$; Connection Failure Distance $\lambda_d$; Graph Feature Smoothness $\lambda_f$; Feature
       Smoothness Hyper-parameter $\rho$
Output: Subgraph $\mathcal{G}_k$
Initiate $\mathcal{G}_k = (\mathcal{V}_k, \mathcal{E}_k)$ with $\mathcal{V}_k = \varnothing$; Set expansion step $m = 0$; Set current expansion nodes set $CS = \varnothing$; Set
  next expansion nodes set $NS = \varnothing$;
Randomly pick the root node $v_r$, add $v_r$ into $\mathcal{V}_k$ and $CS$; while $m \le \lfloor \lambda_d / 2 \rfloor$ do
    for $v_i \in CS$ do
        for $v_j \in \{v_j | e_{v_i,v_j} \in \mathcal{E}, v_j \in \mathcal{V} \setminus \mathcal{V}_k\}$ do
            if $\lambda_{f(v_i,v_j)} \ge \rho\lambda_f$ then
                $\mathcal{V}_k = \mathcal{V}_k \cup \{v_j\}$;
                $\mathcal{E}_k = \mathcal{E}_k \cup \{e_{v_i,v_j}\}$;
                $NS = NS \cup \{v_j\}$;
        end
    end
    $CS = NS$;
    $NS = \varnothing$;
    $m = m + 1$;
end
return $\mathcal{G}_k$;
```

4.3 | Subgraph-based Training in MeGuide

MeGuide can support training most widely used GNN models (e.g., GAT, GCN) to avoid the three problems mentioned above. Different from training GNN models with the full graph $\mathcal{G}$, MeGuide employs the subgraphs sampled from $\mathcal{G}$ by MeGuide Sampler in each training iteration. In this way, a smaller size of coefficient matrices and only part of nodes are loaded into the GNN model during each training iteration. For a specific iteration, a subgraph $\mathcal{G}_t$ is selected from the set of sampled subgraphs $\mathbb{B}$, whose own the ground truth of nodes in $\mathbf{y}_{\mathcal{G}_t}$. The GNN model $H_{\mathbf{W}}(\cdot)$ to be trained is built on $\mathcal{G}_t$ and calculates the loss via forwarding propagation. Then, the weights $\mathbf{W}$ of the GNN model are updated via SGD. After enough iterations, we can achieve the GNN model with trained weights. We describe the training process of MeGuide in detail by the pseudocode in Algorithm 2.

4.4 | Representation Aggregation-based Prediction

After we achieve the GNN model with trained weights, these models can already make predictions for unlabeled nodes (testing samples). When facing the inductive learning in multiple graphs, we can feedforward the multiple unseen graphs in the test set to the GNN model directly. However, for the transductive learning in one single large graph, it is still difficult to feed the full graph into the model and implement the forward propagation to make predictions. Especially the memory space may not be able to load the full graph. In this case, MeGuide continues to use the sampled subgraph to make predictions in a semi-supervised way.

There are two problems in using subgraphs to predict: 1, in the training subgraph set $\mathbb{B}$, may not all unlabeled nodes are included. 2, unlabeled nodes can be sampled into multiple subgraphs, so it is possible to output multiple conflicting prediction results. For problem 1, MeGuide will sample extra subgraphs using the missing unlabeled nodes

Algorithm 2: MeGuide Training for GNNs

```
Input: Graph $\mathcal{G}$; GNNs model $H_{\mathbf{W}}(\cdot)$; loss function $Loss(\cdot)$; subgraph mini-batch size $M$
Output: Trained $H_{\mathbf{W}}(\cdot)$
Initialize training subgraph set $\mathbb{B} = \varnothing$;
for $k = 1, 2, \dots, M$ do
    $\mathcal{G}_k \leftarrow$ MeGuide Sampler;   /* By Algorithm 1 */
    $\mathbb{B} = \mathbb{B} \cup \{\mathcal{G}_k\}$;
end
for each iteration do
    Select a subgraph $\mathcal{G}_t$ from batch;
    GNN model $H_{\mathbf{W}}(\cdot)$ construction on $\mathcal{G}_t$;
    Forward propagation to calculate the loss value: $loss = Loss(H_{\mathbf{W}}(\mathcal{G}_t), \mathbf{y}_{\mathcal{G}_t})$;   /* $\mathbf{y}_{\mathcal{G}_t}$ denotes the ground
      truth of nodes in $\mathcal{G}_t$. */
    Backpropagation to update weights $\mathbf{W}$;
end
return $H_{\mathbf{W}}(\cdot)$
```

of $\mathbb{B}$ as root nodes. These extra subgraphs along with $\mathbb{B}$ will constitute the testing subgraph set $\mathbb{T}$. To deal with problem 2, MeGuide implements aggregation on multiple representations of the same node and trains the predictor (e.g. classification layer) with the aggregated representations of labeled nodes. For example, a labeled node v_i are sampled in $\mathcal{G}_{t_1}, \mathcal{G}_{t_2}, \mathcal{G}_{t_3}$, and we extract the representations $\mathbf{h}_{v_i}^{\mathcal{G}_{t_1}}, \mathbf{h}_{v_i}^{\mathcal{G}_{t_2}}, \mathbf{h}_{v_i}^{\mathcal{G}_{t_3}}$ learned by the GNN model from each subgraph. In this paper, we use the mean aggregator to combine all representations as:

$$\mathbf{h}_{v_i} \leftarrow \mathbf{MEAN}(\{\mathbf{h}_{v_i}^{\mathcal{G}_{t_1}}, \mathbf{h}_{v_i}^{\mathcal{G}_{t_2}}, \mathbf{h}_{v_i}^{\mathcal{G}_{t_3}}\}) \tag{18}$$

The aggregated representation $\mathbf{h}_{v_i}$ will be used to train a new predictor. The aggregation process is the same for unlabeled nodes. The aggregated representations of unlabeled nodes will be used to make predictions by the new predictor. The representations of the same node in different subgraphs essentially embed the content of partial neighboring nodes and topology information of different local parts of the full graph. Therefore, the way to aggregate the representations of the same node in different subgraphs can enable the final representation with more comprehensive information.

5 | EXPERIMENTS

To show the effectiveness and efficiency of MeGuide, extensive experiments have been conducted on benchmark datasets. This section first describes the datasets used in experiments and then introduces the experimental settings in detail. Generally, we aim to answer the following evaluation questions based on experimental results together with the detailed analysis:

- **Question 1**: Can the subgraph-based training of MeGuide effectively train GNN models and overcome the aforementioned three problems?
- **Question 2**: Can the MeGuide Sampler provide powerful subgraphs to support effective and efficient training?

TABLE 1 Statistics of the Datasets in Experiments

	Transductive			Inductive	
	Cora	Citeseer	Pubmed	Flickr	Reddit
# Nodes	2708	3327	19717	89250	232965
# Edges	5429	4732	44338	899756	11606919
# Features	1433	3703	500	500	602
# classes	7	6	3	7	41
Train Rate	0.052	0.036	0.003	0.5	0.66

- **Question 3**: Can the representation aggregation-based prediction of MeGuide improve the performance of original GNN models?

5.1 | Experiment Settings

5.1.1 | Datasets

Five benchmark datasets are used to evaluate the proposed framework MeGuide: Cora, Citeseer, PubMed [43], Flickr, and Reddit [14]. The descriptions of every dataset locate in Table 1. Cora, Citeseer, and Pubmed [43] are standard citation network benchmark datasets. Flickr [14, 44] is built by forming links between images sharing common metadata from Flickr. Edges are formed between images from the same location, submitted to the same gallery, group, or set, images sharing common tags, images taken by friends, etc. Zeng et al. [14] scan over the 81 tags of each image and manually merged them to 7 classes. Each image belongs to one of the 7 classes, which is used as the label for images. Reddit [10] is a graph dataset constructed from Reddit posts. The node label is the community, or "subreddit" that a post belongs to. These datasets involve both the transductive learning task and inductive learning task. The transductive task in our experiments is semi-supervised node classification on one single graph; the inductive task is the node classification on multiple graphs. The statistic information of them is presented in Table 1. The train data rate in the table means the ratio of training data over the full dataset.

5.1.2 | GNN Models for Learning

In our experiments, we adopt two powerful and widely used GNN models, GCN [19] and GAT [17] as the basic models. The settings of these two models are different for various datasets. Specifically, both GCN and GAT contain two layers, and the size of the hidden layer is 32 for Cora, Citeseer, and Pubmed datasets; for Flickr and Reddit datasets, the GCN hidden layer size is 128, and the GAT hidden layer size is 32. For the training of GNNs, we select a part of node data as the validation set: for Cora, Citeseer, and Pubmed datasets, each of the validation sets includes 500 nodes, and the test set contains 1000 nodes; for Flickr and Reddit datasets, two percent of all nodes are included as validation sets, and another ten thousand nodes are selected into the test sets. We set the dropout rate as 0.5 and adopt the Adam [45] as the optimizer for back-propagation. The learning rate is 0.01, and the weight decay rate is 5×10^{-4}.

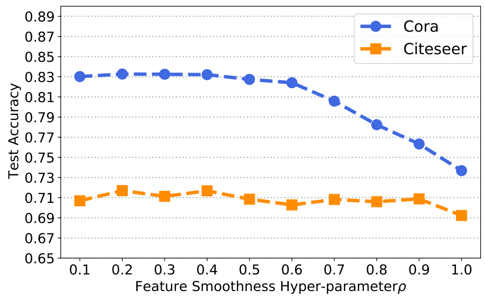

FIGURE 3 MeGuide$_{GCN}$ Results with Different Feature Smoothness Hyper-parameter ρ.

5.1.3 | Comparison Methods

We compare the proposed MeGuide with several state-of-the-art baseline methods, including vanilla GCN, GAT, GraphSAGE, Cluster-GCN, GraphSAINT, and RippleWalk (RWT). These comparison methods can be divided into two categories: (1) full graph-based methods, which use the full graph in each training iteration; (2) subgraph-based methods involve only a subgraph in each train iteration. GCN and GAT belong to the first category for our selected baseline methods, while Cluster-GCN, GraphSAINT, and RWT are in the second category. A detailed description of the comparison methods is listed as follow:

Full Graph-based Methods

- **GCN** [19]: GCN is a semi-supervised method proposed for the node classification task. The input of GCN is the full graph.
- **GAT** [17]: GAT is an attention-based graph neural network for the node classification. GAT operates on the full graph.
- **GraphSAGE** [10]: GraphSAGE is a general inductive framework that leverages node feature information to efficiently generate node embeddings for previously unseen data. GraphSAGE does not require that all nodes in the graph are present during training.

Subgraph-based Methods

- **MeGuide**: MeGuide is the general learning framework for GNN models proposed in this paper.
- **Cluster-GCN** [6]: Cluster-GCN conducts subgraph sampling based on the graph clustering, and leverages the sampled subgraphs from different clusters to train the GNNs.
- **GraphSAINT** [14]: GraphSAINT is designed for inductive learning on graph datasets. It samples subgraphs with different types of random samplers and uses these subgraphs to constitute the mini-batch to train GNN structures.
- **RippleWalk** [4]: RippleWalk (RWT) is a general training framework for GNN models on both transductive and inductive learning tasks. It samples subgraphs with the expansion steps from initial nodes, and return subgraphs with pre-defined target sizes. RWT constructs the mini-batch with sampled subgraphs to train GNN models.

Generally, the key of subgraph based methods is how to design the subgraph samplers. Apart from the subgraph-based methods mentioned above, we also compare MeGuide with the following subgraph sampling methods:

TABLE 2 Test Accuracy Results on All Datasets

Methods	Transductive			Inductive	
	Cora	Citeseer	Pubmed	Flickr	Reddit
GraphSAGE	0.8096 ±0.0103	0.6772±0.0110	0.7589±0.0161	0.4337±0.0201	0.9353±0.0211
Cluster-GCN	0.6820±0.0638	0.6280±0.0430	0.7947±0.0036	0.4097±0.0405	**0.9523±0.0338**
GraphSAINT	0.8045±0.0114	0.6961±0.0299	0.7424±0.0202	0.4848±0.0252	0.9451±0.0062
GCN	0.8150±0.0050	0.7030±0.0050	0.7890±0.0070	0.4400±0.0388	0.9333±0.0147
GCN + Random	0.7945±0.0105	0.687±0.0133	0.7345±0.0163	0.4713±0.0083	0.8243±0.0336
GCN + BFS	0.8144±0.0045	0.7079±0.0106	0.7971±0.0021	0.4754±0.0046	0.8123±0.0434
GCN + RWT	0.8250±0.0016	0.7127±0.0018	0.8259±0.0225	0.4797±0.0038	0.9385±0.0309
MeGuide$_{GCN}$	**0.8327±0.0130**	**0.7170±0.0252**	**0.8317±0.0167**	**0.5189±0.0121**	**0.9499±0.0232**
GAT	**0.8300±0.0070**	0.7130±0.0082	0.7903±0.0033	-	-
GAT + Random	0.7921±0.0236	0.6607±0.0376	0.6765±0.0415	0.4534±0.0040	0.6452±0.0447
GAT + BFS	0.7756±0.0281	0.6500±0.0493	0.7080±0.0511	0.4642±0.0232	0.7297±0.0403
GAT + RWT	0.7994±0.0309	0.7212±0.0142	**0.8210±0.0019**	0.4724±0.0089	0.8699±0.0179
MeGuide$_{GAT}$	0.8017±0.0110	**0.7250±0.0121**	0.8174±0.0275	**0.4808±0.0301**	**0.8776±0.0320**

"-" insufficient memory.

- **Random**: Random sampler generates subgraph by randomly select nodes from the full graph to a pre-defined target subgraph size.
- **BFS**: BFS sampler conducts the subgraph sampling from one initial node to expand with BFS strategy, and stop when the expanded subgraph reaches the target size.

We replace MeGuide Sampler with these two samplers in order to make comparisons. MeGuide$_{GAT}$ denotes applying MeGuide on the model GAT. GAT + BFS represents that we test MeGuide on GAT with the setting of replacing MeGuide Sampler with BFS.

5.2 | Experiment Environment

We run the experiments on the Server with 3 GTX-1080 ti GPUs. Codes are implemented in Pytorch 1.4.0, torch-geometric 1.6.0, cudatoolkit 10.1.243, and scikit-learn 0.23.1. Code is available at the github link: https://github.com/YuxiangRen/Measuring-and-Sampling-A-Metric-guided-Subgraph-Learning-Framework-for-Graph-Neural-Network.

5.3 | Experimental Results with Analysis

5.3.1 | Overall performance analysis

The results of both the transductive learning task and the inductive learning task are exhibited in Table 2. We present the accuracy of node classification on the test set. For the baseline methods evaluating the performance with F1

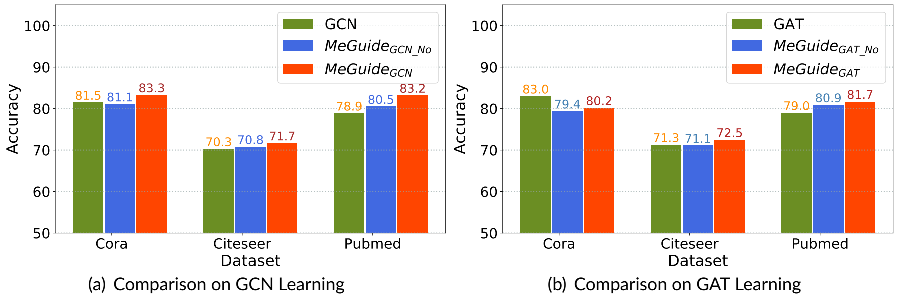

FIGURE 4 Ablation Study of Representation Aggregation-based Prediction.

score in their original papers, out of the need for a consistent evaluation metric, we reimplement their models and run the experiments to report the results based on the metric accuracy. From Table 2, we can observe that MeGuide achieves the best results on 4 out of the 5 datasets, for cases when GCN and GAT as the original GNN models. Regarding the dataset (Reddit) that MeGuide does not obtain the highest accuracy, MeGuide's performance is still competitive compared to the state-of-the-art methods. Therefore, to the **Question 1** we can confidently answer that MeGuide can guarantee the performance of the original GNN models while even help improves the performance of the GNNs. Apart from this, comparing to other subgraph-based methods, including Cluster-GCN, GraphSAINT, and RWT, MeGuide acquires the best results on almost all datasets. This comparison also verifies the effectiveness in GNN model learning, which also illustrates the ability of MeGuide to overcome the three problems encountered by GNN models from the perspective of the final result. In the following part, we will continue to discuss the performance of MeGuide in dealing with the three problems separately.

To answer to the **Question 2**, we can compare the performance among different samplers. The improvement from MeGuide sampler can reach up to 10 percent compared to Random and BFS sampler (e.g., on Pubmed dataset) and 4 percent against RWT (e.g., on Flickr dataset). Such a phenomenon illustrates that the subgraphs involved by MeGuide Sampler are powerful and be able to support effective training for GNN models.

To answer to the **Question 3**, we conducted a set of ablation studies. We remove the representation aggregation-based prediction in MeGuide and directly input the full graph to the trained GNN models for prediction. The experimental results are shown in Figure 4, where $MeGuide_{GCN\_No}$ represents that applying MeGuide without the representation aggregation-based prediction to GCN. It can be seen from the comparison between GCN and $MeGuide_{GCN\_No}$ that the prediction strategy based on the subgraph we designed can improve the performance of trained GNN models. The result is consistent for the comparison between GAT and $MeGuide_{GAT\_No}$. Besides, the comparison between GCN and $MeGuide_{GCN\_No}$ and the comparison between GAT and $MeGuide_{GAT\_No}$ in the Figure 4 can further demonstrate that MeGuide can better train the original GNN models, and this performance improvement does not come from the representation aggregation-based prediction phase.

5.3.2 | Space-consuming Analysis

In each experiment, we record the memory usage of our device and show the values in Figure 5. From the figures, we can see that when using GCN as the original model, on the Cora and Citeseer dataset, the memory usage of MeGuide is smaller than GCN and RWT by a little gap. GCN requires much more memory space than subgraph-based methods when it comes to relatively larger datasets (e.g., Pubmed, Flicker, and Reddit). Such a phenomenon is

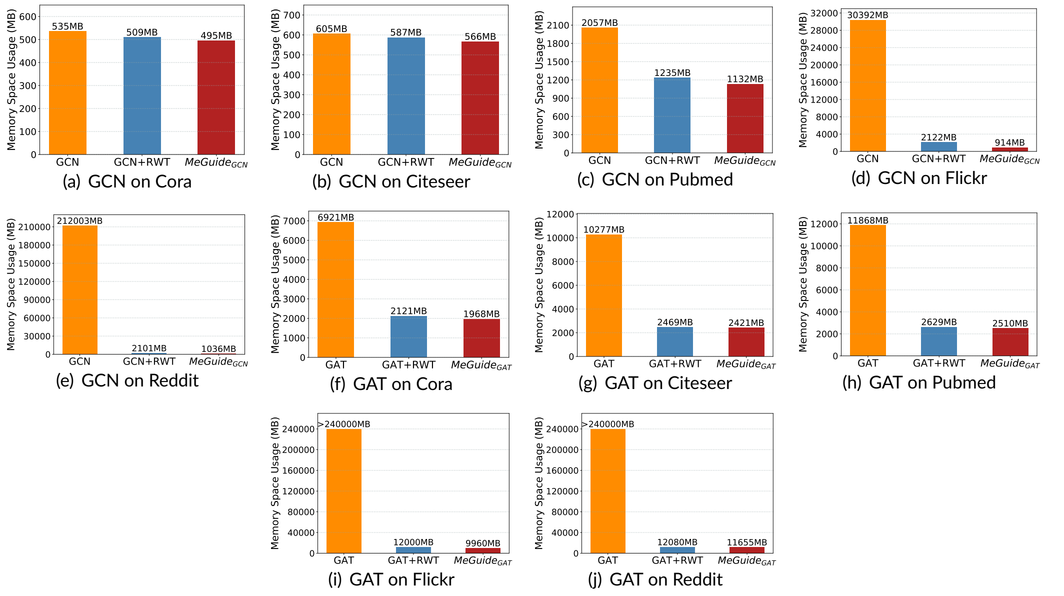

FIGURE 5 Memory Space Usage (The unit is MB) on Different Datasets.

reasonable according to the mechanism of GNNs layer computation. Each layer of GNNs computing loads the current graph's adjacency matrix, the size of which will increase quadratically along with the increasing of graph size (i.e., the number of nodes). A similar phenomenon also occurs when GAT becomes the original model. Generally, compared to full graph-based methods, the advantage of MeGuide is prominent concerning the memory cost. When comparing to RWT, MeGuide keeps at the same level with RWT and has a slight advantage. Therefore, for the **Question 1**, MeGuide successfully break through the memory bottleneck brought by *node dependence* and *neighbors explosion* (mentioned in Section 1).

5.3.3 | Time-consuming Analysis

In the training process, we record the convergence time of each experiment and list the results in Table 3. From the results, it is obvious that subgraph-based methods such as RWT and MeGuide require much less convergence time than the full graph-based methods GCN and GAT. Such fast training convergence speed of subgraph-based methods is easy to understand. Since for the feedforward and backpropagation processes of GNNs computing, subgraph-based methods (e.g., MeGuide) only involve subgraphs instead of the full graph, which limits the occurrence of *node dependence* and *neighbors explosion* to the scope of the subgraph. Therefore, the computation complexity in each training iteration for subgraph-based methods is much lower than full graph-based methods. Besides, compared to RWT, the convergence of MeGuide is also slightly faster, but MeGuide can achieve better performance, which reflects the effectiveness and efficiency of the subgraph sampling by MeGuide.

TABLE 3 Convergence Time (The unit is second)

	Cora	Citeseer	Pubmed	Flickr	Reddit
GCN	4.573	1.968	61.90	1161.92	25370
GCN + RWT	1.964	1.826	8.698	1.179	7.722
MeGuide$_{GCN}$	**1.784**	**1.653**	**5.112**	**1.149**	**7.262**
GAT	413.3	500.1	-	-	-
GAT + RWT	71.44	47.06	139.4	68.06	2614
MeGuide$_{GAT}$	**63.75**	**36.88**	**109.7**	**58.30**	**2367**

"-" insufficient memory for GPU.

TABLE 4 Metrics Value on All Datasets

	Cora	Citeseer	Pubmed	Flickr	Reddit
λ_f	0.123	0.051	0.058	1002.042	775.54
λ_d	8.8	6.5	6.9	5.8	3.9

5.4 | Parameter Analysis

There are some key parameters and metric values employed by our proposed MeGuide. We list the feature smoothness values λ_f and connection failure distance λ_d of all datasets in Table 4. On Cora, Citeseer, and Pubmed datasets, the feature smoothness value is relatively small, illustrating that the nodes in the whole graph share more similar features. The subgraph sampling strategy of MeGuide is based on the values in the table. There is another important hyper-parameter, the feature smoothness hyper-parameter ρ, which determines the expansion condition of MeGuide sampler. We tune different values of ρ and exhibit the results in Figure 3. The trend of results indicates that MeGuide achieves the highest performance when ρ locates in the range of 0.2 to 0.5. When ρ is too small (e.g., $\rho = 0.1$) or becomes larger, the performance worsens. Such a phenomenon matches our expectation, because on the one hand, when ρ is too small, MeGuide is not able to filter nodes with less information gain during the sampling process of MeGuide sampler. On the other hand, when ρ is too large, the expansion step of MeGuide sampler cannot work anymore: only a few neighboring nodes of the already selected nodes are qualified for being sampled.

6 | CONCLUSION

In this paper, we propose a general framework MeGuide for optimizing the training and prediction of GNN models. In MeGuide, we design the subgraph-based training to deal with three non-trivial problems (*neighbors explosion*, *node dependence*, and *oversmoothing*) bothering many GNN models. Different from the existing subgraph-based training methods, we define two metrics that can be used to measure the performance gain of GNN models to guide the sampling of subgraphs. The training performance of the GNN models can be improved through more effective subgraphs. In addition, for the case of the memory bottleneck when using trained GNN models to predict on a single large graph, MeGuide provides a solution for prediction based on subgraphs, which is an unsolved problem left by existing subgraph-based training methods. Extensive experiments on 5 benchmark graph datasets and 2 widely used GNN models demonstrate the effectiveness of MeGuide, where GNN models not only achieve the comparative or even better performance but less training time and device memory space are required.

references

[1] Zhang J. Social network fusion and mining: a survey. arXiv preprint arXiv:180409874 2018;.

[2] Ren Y, Zhang J. HGAT: Hierarchical Graph Attention Network for Fake News Detection. arXiv 2020;p. arXiv–2002.

[3] Wang Q, Mao Z, Wang B, Guo L. Knowledge graph embedding: A survey of approaches and applications. IEEE Transactions on Knowledge and Data Engineering 2017;29(12):2724–2743.

[4] Bai J, Ren Y, Zhang J. Ripple Walk Training: A Subgraph-based training framework for Large and Deep Graph Neural Network. In: IJCNN; 2021. .

[5] Xu K, Hu W, Leskovec J, Jegelka S. How powerful are graph neural networks? In: ICLR; 2019. .

[6] Chiang WL, Liu X, Si S, Li Y, Bengio S, Hsieh CJ. Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks. In: KDD; 2019. .

[7] Zhao L, Akoglu L. PairNorm: Tackling Oversmoothing in GNNs. arXiv:190912223 2019;.

[8] Li Q, Han Z, Wu XM. Deeper insights into graph convolutional networks for semi-supervised learning. In: AAAI; 2018. .

[9] Rong Y, Huang W, Xu T, Huang J. DropEdge: Towards the Very Deep Graph Convolutional Networks for Node Classification. In: arXiv:1907.10903; 2019. .

[10] Hamilton W, Ying Z, Leskovec J. Inductive representation learning on large graphs. In: NIPS; 2017. .

[11] Chen J, Ma T, Xiao C. Fastgcn: fast learning with graph convolutional networks via importance sampling. arXiv:180110247 2018;.

[12] Chen J, Zhu J, Song L. Stochastic training of graph convolutional networks with variance reduction. In: ICML; 2018. .

[13] Zou D, Hu Z, Wang Y, Jiang S, Sun Y, Gu Q. Layer-Dependent Importance Sampling for Training Deep and Large Graph Convolutional Networks. In: NeurIPS; 2019. .

[14] Zeng H, Zhou H, Srivastava A, Kannan R, Prasanna V. Graphsaint: Graph sampling based inductive learning method. arXiv preprint arXiv:190704931 2019;.

[15] Hou Y, Zhang J, Cheng J, Ma K, Ma RT, Chen H, et al. Measuring and improving the use of graph information in graph neural networks. In: International Conference on Learning Representations; 2019. .

[16] Bruna J, Zaremba W, Szlam A, LeCun Y. Spectral networks and locally connected networks on graphs. arXiv:13126203 2013;.

[17] Veličković P, Cucurull G, Casanova A, Romero A, Liò P, Bengio Y. Graph Attention Networks. In: ICLR; 2018. .

[18] Monti F, Boscaini D, Masci J, Rodola E, Svoboda J, Bronstein MM. Geometric deep learning on graphs and manifolds using mixture model cnns. In: CVPR; 2017. .

[19] Kipf TN, Welling M. Semi-supervised classification with graph convolutional networks. In: ICLR; 2017. .

[20] Xinyi Z, Chen L. Capsule graph neural network. In: International conference on learning representations; 2018. .

[21] Sun FY, Hoffmann J, Verma V, Tang J. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. arXiv preprint arXiv:190801000 2019;.

[22] Ying Z, You J, Morris C, Ren X, Hamilton W, Leskovec J. Hierarchical graph representation learning with differentiable pooling. In: Advances in neural information processing systems; 2018. p. 4800–4810.

[23] Ren Y, Bai J, Zhang J. Label Contrastive Coding based Graph Neural Network for Graph Classification. In: Database Systems for Advanced Applications Springer International Publishing; 2021. p. 123–140.

[24] Zhou J, Cui G, Hu S, Zhang Z, Yang C, Liu Z, et al. Graph neural networks: A review of methods and applications. AI Open 2020;1:57–81.

[25] Wu Z, Pan S, Chen F, Long G, Zhang C, Philip SY. A comprehensive survey on graph neural networks. IEEE transactions on neural networks and learning systems 2020;32(1):4–24.

[26] Defferrard M, Bresson X, Vandergheynst P. Convolutional neural networks on graphs with fast localized spectral filtering. In: NIPS; 2016. .

[27] Levie R, Monti F, Bresson X, Bronstein MM. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. IEEE Transactions on Signal Processing 2018;.

[28] Liao R, Zhao Z, Urtasun R, Zemel RS. Lanczosnet: Multi-scale deep graph convolutional networks. arXiv:190101484 2019;.

[29] Henaff M, Bruna J, LeCun Y. Deep convolutional networks on graph-structured data. arXiv:150605163 2015;.

[30] Li R, Wang S, Zhu F, Huang J. Adaptive graph convolutional neural networks. In: AAAI; 2018. .

[31] Ying R, He R, Chen K, Eksombatchai P, Hamilton WL, Leskovec J. Graph convolutional neural networks for web-scale recommender systems. In: KDD; 2018. .

[32] Gao H, Wang Z, Ji S. Large-scale learnable graph convolutional networks. In: KDD; 2018. .

[33] Xu K, Li C, Tian Y, Sonobe T, Kawarabayashi Ki, Jegelka S. Representation Learning on Graphs with Jumping Knowledge Networks. In: ICML; 2018. .

[34] Lee JB, Rossi RA, Kong X, Kim S, Koh E, Rao A. Graph Convolutional Networks with Motif-based Attention. In: CIKM; 2019. .

[35] Klicpera J, Bojchevski A, Günnemann S. Predict then Propagate: Graph Neural Networks meet Personalized PageRank 2019;.

[36] Haonan L, Huang SH, Ye T, Xiuyan G. Graph star net for generalized multi-task learning. arXiv:190612330 2019;.

[37] Abu-El-Haija S, Perozzi B, Kapoor A. Mixhop: Higher-order graph convolution architectures via sparsified neighborhood mixing. arXiv:190500067 2019;.

[38] Chen M, Wei Z, Ding B, Li Y, Yuan Y, Du X, et al. Scalable graph neural networks via bidirectional propagation. arXiv preprint arXiv:201015421 2020;.

[39] Zhang J, Meng L. GResNet: Graph Residual Network for Reviving Deep GNNs from Suspended Animation. In: arXiv:1909.05729; 2019. .

[40] Kullback S, Leibler RA. On information and sufficiency. The annals of mathematical statistics 1951;22(1):79–86.

[41] Zhang J, Meng L. GResNet: Graph Residual Network for Reviving Deep GNNs from Suspended Animation. arXiv preprint arXiv:190905729 2019;.

[42] Huang W, Rong Y, Xu T, Sun F, Huang J. Tackling Over-Smoothing for General Graph Convolutional Networks. arXiv preprint arXiv:200809864 2020;.

[43] Sen P, Namata G, Bilgic M, Getoor L, Galligher B, Eliassi-Rad T. Collective classification in network data. AI magazine 2008;.

[44] McAuley J, Leskovec J. Image labeling on a network: using social-network metadata for image classification. In: European conference on computer vision Springer; 2012. p. 828–841.

[45] Kingma DP, Ba JL. ADAM: A method for stochastic optimization. In: ICLR; 2015. .

Jiyang Bai received the bachelor degree in information and numerical science from Nankai University, China, in 2018. He is pursuing a PhD degree in the Department of Computer Science at the Florida State University. His main research areas are data mining and machine learning, especially focus on the graph mining, graph neural networks, graph similarity search.

Yuxiang Ren received the bachelor degree in software engineering and the bachelor degree in law from Nanjing University, China, in 2015, and the Ph.D. degree in Computer Science from Florida State University in 2021. His main research areas are data mining and machine learning, especially focus on the development and analysis of algorithms for social and information networks, as well as heterogeneous graph mining and fake news detection.

Jiawei Zhang received the bachelor's degree in computer science from Nanjing University, China, in 2012, and the Ph.D. degree in computer science from the University of Illinois at Chicago, USA, in 2017. He has been an Assistant Professor with the Department of Computer Science, Florida State University, Tallahassee, FL, USA, since 2017. He founded IFM Lab in 2017, and has been working as the director since then. IFM Lab is a research oriented academic laboratory, providing the latest information on fusion learning and data mining research works and application tools to both academia and industry.