Graph Cross Supervised Learning via Generalized Knowledge

Xiangchi Yuan
Brandeis University
Waltham, MA, USA
Georgia Institute of Technology
Atlanta, GA, USA
xiangchiyuan@brandeis.edu

Yijun Tian
University of Notre Dame
South Bend, IN, USA
yijun.tian@nd.edu

Chunhui Zhang
Dartmouth College
Hanover, NH, USA
chunhui.zhang.gr@dartmouth.edu

Yanfang Ye
University of Notre Dame
South Bend, IN, USA
yye7@nd.edu

Nitesh V. Chawla
University of Notre Dame
South Bend, IN, USA
nchawla@nd.edu

Chuxu Zhang
Brandeis University
Waltham, MA, USA
chuxuzhang@brandeis.edu

ABSTRACT

The success of GNNs highly relies on the accurate labeling of data. Existing methods of ensuring accurate labels, such as weakly-supervised learning, mainly focus on the existing nodes in the graphs. However, in reality, new nodes always continuously emerge on dynamic graphs, with different categories and even label noises. To this end, we formulate a new problem, *Graph Cross-Supervised Learning*, or *Graph Weak-Shot Learning*, that describes the challenges of modeling new nodes with novel classes and potential label noises. To solve this problem, we propose **Li**pshitz-regularized **M**ixture-of-**E**xperts similarity network (LIME), a novel framework to encode new nodes and handle label noises. Specifically, we first design a node similarity network to capture the knowledge from the original classes, aiming to obtain insights for the emerging novel classes. Then, to enhance the similarity network's generalization to new nodes that could have a distribution shift, we employ the Mixture-of-Experts technique to increase the generalization of knowledge learned by the similarity network. To further avoid losing generalization ability during training, we introduce the Lipschitz bound to stabilize model output and alleviate the distribution shift issue. Empirical experiments validate LIME's effectiveness: we observe a substantial enhancement of up to 11.34% in node classification accuracy compared to the backbone model when subjected to the challenges of label noise on novel classes across five benchmark datasets. The code can be accessed through https://github.com/xiangchi-yuan/Graph-Cross-Supervised-Learning.

CCS CONCEPTS

• **Computing methodologies → Neural networks**; • **Information systems → Social networks**.

KEYWORDS

Graph Cross-Supervised Learning; Graph Weak-Shot Learning; Mixture-of-Experts; Lipschitz Constant; Label Noise

KDD '24, August 25–29, 2024, Barcelona, Spain
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0490-1/24/08.
https://doi.org/10.1145/3637528.3671830

ACM Reference Format:
Xiangchi Yuan, Yijun Tian, Chunhui Zhang, Yanfang Ye, Nitesh V. Chawla, and Chuxu Zhang. 2024. Graph Cross Supervised Learning via Generalized Knowledge. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24), August 25–29, 2024, Barcelona, Spain.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3637528.3671830

1 INTRODUCTION

Graph Neural Networks (GNNs) employ message-passing mechanisms to iteratively update node representations, allowing them to aggregate information from neighboring nodes. This capability makes GNNs particularly potent for modeling and learning from graph-structured data. Consequently, GNNs have consistently demonstrated state-of-the-art performance across various graph-related tasks, such as node classification [9, 16, 34] and graph classification [40, 45, 46]. Furthermore, GNNs have proven to be notably effective in real-world applications, including social network analysis [23, 55], fraud detection [56], natural language processing [6, 33, 47], and recommendation systems [5, 12, 48].

However, the successes of GNNs rely heavily on the precise annotation of training data. If nodes are labeled with noise, the model can be fooled, leading to inaccurate representations with nodes belonging to different categories cross together. Considering there are always new nodes with noisy or sparse labels added to the graphs, encoding nodes on real-world web graphs such as social, citation, and sales networks can be challenging. Therefore, many graph weakly supervised learning methods are proposed to handle label noise [4, 24, 31]. For example, NRGNN [4] and RTGNN [27] are proposed to predict accurate pseudo-labels and distinguish noisy labels from clean labels to provide supervision for the new nodes with label noise. Nevertheless, these methods are limited to handling new nodes that have pre-existing classes in the training set. When there is a need to model the nodes with novel classes and potential label noises, these methods can not fully utilize the knowledge contained in the original graph and fail to perform well.

Although predicting new nodes that have existing classes can be straightforward, it is challenging to learn and predict the new nodes with classes that are distinct from existing node classes, especially when these nodes have label noise. In light of this, we introduce a new problem, i.e. **Graph Cross-Supervised (Weak-Shot) Learning**, which corresponds to learning from existing nodes

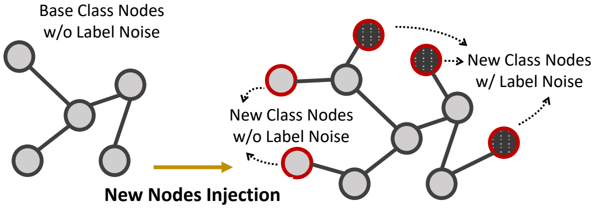

Figure 1: Graph Cross-Supervised Learning: in the original graph, base class nodes with accurate labels; then new novel class nodes with label noise emerge on the original graph.

with correct labels to deduce the true category of new nodes, while noises can present for these new nodes.

This problem is realistic and important (detailed discussion on this bitter lesson is provided Appendix D): experts can annotate nodes on small-scale graphs with high quality at the start, while it is unrealistic to continually annotate new nodes with novel labels accurately when tons of these nodes continuously emerge, considering limited annotation resources. Therefore, it is common and natural to annotate nodes through cheap and easy-maintaining approaches like crowdsourcing and pseudo labels by clustering, which bring label noise. To solve this problem, we propose a novel framework called LIME (**Li**pshitz-regularized **M**ixture-of-**E**xperts similarity network). Specifically, we first obtain the position features and concatenate them to node features. Then the concatenated features of the original graph are used to train a similarity network, which encodes the base class node without label noise and generates similarity scores of node embedding pairs. After that, the similarity network inferences on new class nodes with potential label noises. For each node, the similarity scores between it and the other nodes with the same label are averaged as its loss weight during subsequent model training for node classification. To generalize the similarity knowledge from training node pairs to inference node pairs, we utilize the Mixture-of-Experts technique to increase the generalization of the similarity network and introduce the Lipshitz constant as loss regularization with theoretical analysis. The major contributions of this paper are summarized as follows:

- To the best of our knowledge, this work first proposes a new realistic graph learning scenario called **Graph Cross-Supervised Learning**. According to our empirical studies in the experimental section, most current works are ineffective in this setting because they do not fully utilize existing labels of the base class nodes.
- To solve the problem, we propose LIME model that utilizes a similarity network to extract implicit knowledge for weighing the training loss of new class nodes with label noises. To further generalize this process, the Mixture-of-Experts module and the corresponding Lipshitz bound are proposed.
- We build benchmark datasets and related dataset synthetic tools for the new problem. Extensive experiments on the new datasets demonstrate the effectiveness of our method. Results show that LIME outperforms popular baselines and handles the graph cross-supervised learning scenario well against label noises.

2 RELATED WORK

Graph Neural Networks (GNNs). GNNs have gained widespread attention due to their ability to effectively learn non-Euclidean data and achieve outstanding performance in various graph mining tasks [1, 3, 9, 32, 44, 53]. Early works of GNNs such as graph convolutional networks (GCN) are proposed to apply convolutional operations on graph data [7, 16, 42]. Graph attention networks were introduced to improve GCNs by incorporating attention mechanisms to weigh the importance of neighboring nodes during message passing [34, 39]. In addition, graph recurrent neural networks have been proposed to address the limitations of GNNs in handling long-distance message passing on large graphs by incorporating gating mechanisms inspired by recurrent neural networks [28]. To overcome the problem of oversmoothing, deeper GNNs were constructed using skip connections to learn more comprehensive representations [18–20]. While prior works have primarily focused on improving standard accuracy, our method addresses the growing concerns of label noise and novel node classes.

Graph Learning with Label Noise. Previous research [24, 54] has shown that deep graph models are vulnerable to label noises, which urges the need to design robust graph learning methods against label noise. Among current methods that handle node label noise, D-GNN [24] employs backward loss correction [26] to improve performance. NRGNN [4] learns robust node representations with noisy and sparse labels by connecting unlabeled nodes with labeled nodes and further predicting accurate pseudo-labels to provide supervision. Different from NRGNN which explicitly governs noisy labels, RTGNN [27] distinguishes noisy labels from clean labels and provides label correction to reduce the impact of label noises. However, when there is the need to model new nodes with novel classes and potential label noises, these methods do not utilize knowledge contained by strongly labeled base class nodes in the original graph to handle label noise with novel classes.

Cross-Supervised (Weak-Shot) Learning. Cross-Supervised learning refers to learning novel categories with cheap weak labels, with knowledge from a set of base categories that are already accurately labeled. Machine learning and computer vision researchers have explored similar settings (also named mixed-supervised learning or weak-shot learning) in different tasks, such as object detection [10, 21, 58], fine-grained classification [2], semantic segmentation [58], and instance segmentation [11, 17]. To the best of our knowledge, we are the first to attempt to define cross-supervised learning on graphs.

3 PRELIMINARY

Problem Definition. Consider an undirected attributed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{X}, \mathcal{Y})$, where $\mathcal{V}$ is the set of N nodes, $\mathcal{E}$ is the set of edges, $\mathcal{X} \in \mathbb{R}^{N \times D}$ denotes the set of node features with D dimensions, and $\mathcal{Y}$ denotes node labels with $\mathcal{K}$ classes. The label for these nodes with **base classes** is **clean**. After new nodes with **novel classes** emerge on original graph, the graph becomes to $\mathcal{G}' = (\mathcal{V}', \mathcal{E}', \mathcal{X}', \mathcal{Y}')$. Note that $\mathcal{Y}'$ has $\mathcal{K}'$ classes with $|\mathcal{K}' - \mathcal{K}|$ new classes annotated with **noise**, and the ground-truth set is denoted as $\mathcal{Y}^T$. The objective is to learn a model f_θ to maximum prediction accuracy in terms of the ground-truth label. The Graph Cross-Supervised Learning objective can be formulated as:

$$\min_{\theta} \mathcal{L}(f_\theta(\mathcal{G}'), \mathcal{Y}^T). \tag{1}$$

Sparse Mixture of Experts [29]. The Mixture-of-Experts (MoE) ensembles multiple expert models to make predictions or perform other tasks. The basic idea behind MoE is to divide the input space into numerous partitions and assign different experts to different partitions. Each expert is a specialized model that is trained to perform well on a specific subset of the input space. The final output is obtained by assembling the predictions of all the experts, typically using a gating mechanism that determines the weight of each expert based on the input. Formally, let h be the input space to the MoE, and $\mathbf{E} = \{E_i(\cdot)\}_{i=1}^{N}$ denotes that an MoE layer consists of N experts. The output of the MoE is given by:

$$y = \sum_{i=1}^{N} p_i(h) E_i(h), \tag{2}$$

where E_i is the i-th expert model, $p_i(h)$ is the weight assigned to the i-th expert for the input space h. The weights are typically obtained from a gating network, which is trained to assign higher weights to experts that are more likely to make accurate predictions for a given input space. A popular approach to design the gating mechanism is to use a softmax function to calculate the weights for gating, and to use neural networks as the expert models by $p_i(h) = \frac{\exp(t(h)_i)}{\sum_{j=1}^{N}\exp(t(h)_j)}$, where $t(h)$ is a linear transformation to compute the logits of the experts given the input space h, $t(h)_i$ is the i-th value of the obtained logits, which is the weight for the i-th expert in the current layer. Previous graph MoE models are designed to improve fairness [22], robustness [41, 49, 50, 52], and their practical use to real-world [25, 51]. Different from them, we utilize MoE to improve the generalization of node pairs in this work.

Lipschitz Constant [14]. A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is established to be Lipschitz continuous on an input set $\mathcal{X} \subseteq \mathbb{R}^n$ if there exists a constant $K \geq 0$ such that for all $x, y \in \mathcal{X}$, f satisfies the following inequality:

$$||f(x) - f(y)|| \leq K||x - y||, \forall x, y \in \mathcal{X}. \tag{3}$$

The smallest possible K in Equation (3) is the Lipschitz constant of f, denoted as $\mathrm{Lip}(f)$:

$$\mathrm{Lip}(f) = \sup_{x,y\in\mathcal{X}, x\neq y} \frac{||f(x) - f(y)||}{||x - y||}, \tag{4}$$

and we say that f is a K-Lipschitz function. Previous works such as [35] propose to use Lipschitz bound as spectral norm regularization to make the model more stable and [13] improve model fairness by using Lipschitz regularizer.

4 METHOD

This section presents our proposed LIME model to increase the GNN performance when novel category data has label noise. Figure 2 illustrates the overall design of the model. LIME consists of two training processes. First, a Mixture-of-Experts Similarity Network (**SimMoE**) is introduced and trained on the original graph to predict whether two nodes are similar or not in terms of their labels. SimMoE consists of two key modules: Mixture-of-Experts GNN encoder (**GNNMoE**) and Mixture-of-Experts MLP similarity predictor (**MLPMoE**). The pipeline of SimMoE can be divided into three steps: concatenating positional embedding, encoding node features by GNNMoE, and predicting similarity scores by MLPMoE. In addition, we further introduce the Lipshitz bound to regularize the model to generalize learned knowledge and avoid overfitting. Next, we utilize the trained SimMoE to infer the graph with new nodes and calculate accumulated similarity scores between nodes that have the same label. Since outlier nodes with noise labels are dissimilar to the nodes with accurate labels, they have lower accumulated similarity scores. Finally, the similarity scores are taken as the weights of node losses to enforce model discard knowledge from nodes with wrong labels during the training process. The details of these steps are described in the following section.

4.1 Training the Mixture-of-Experts Similarity Network on the Original Graph

SimMoE which contains GNNMoE and MLPMoE is used to predict whether two nodes are similar or not. The illustration of SimMoE is shown in part (a) of Figure 2. Specifically, input node features of the original graph $\mathcal{G}$ are $\boldsymbol{X} \in \mathbb{R}^{N\times D}$, where N is the number of nodes and D is the feature dimension. $\boldsymbol{X}$ are concatenated to the positional features to obtain final node features $\boldsymbol{X_F} \in \mathbb{R}^{N\times D'}$, where D' is the final node feature dimension. Second, the GNNMoE encodes features to embeddings $\boldsymbol{Z} \in \mathbb{R}^{N\times H}$, where H is the hidden dimension. After that, we pair the node embeddings to obtain node pair similarity features $\boldsymbol{E} \in \mathbb{R}^{N_t\times 2H}$, where N_t is the number of similarity features. Finally, the node similarity features are used to train the MLPMoE with binary similarity labels, i.e., “similar" (two nodes have the same label) and “dissimilar" (two nodes have different labels).

Concatenating Positional Embeddings. To comprehensively learn the positional information of the subgraph, we propose to extend node features by obtaining positional embeddings from unsupervised positional learning methods such as Node2Vec [8]. Specifically, the Node2Vec model learns the positional information from the adjacency matrix $\boldsymbol{A}$, and the learned embeddings $\boldsymbol{X_P}$ of nodes is concatenated to original node features $\boldsymbol{X}$ to obtain final node features $\boldsymbol{X_F}$. This process can be formulated as:

$$\boldsymbol{X_P} = \mathrm{Node2Vec}(\boldsymbol{A}), \boldsymbol{X_F} = \mathrm{CONCAT}(\boldsymbol{X_P}, \boldsymbol{X}). \tag{5}$$

Encoding Node Features by GNNMoE. When we employ the SimMoE to infer the new graph with injected new nodes, there exists a distribution shift since the model is trained on the original graph with base classes. To improve the generalization of knowledge learned by SimMoE under this condition, we incorporate the MoE techniques in SimMoE to learn diverse representations. SimMoE learns rich representations from diverse experts and during the inference, the gate network distributes each input feature to their most suitable expert with the best generalization to the corresponding input. Specifically, we divide FC-layer Linear$(\cdot)$ into multiple expert networks to process the target representation h. The process is formulated as follows:

$$\mathrm{LinMoE}(h) = \sum_{i\in\mathcal{T}} p_i(h) \cdot \mathrm{Linear}_i(h), \tag{6}$$

where $\mathcal{T}$ represents the set of activated top-k expert indices. LinMoE$(\cdot)$ combines the output of multiple expert networks. In particular, the top-k activated expert indices are determined by the gate-value

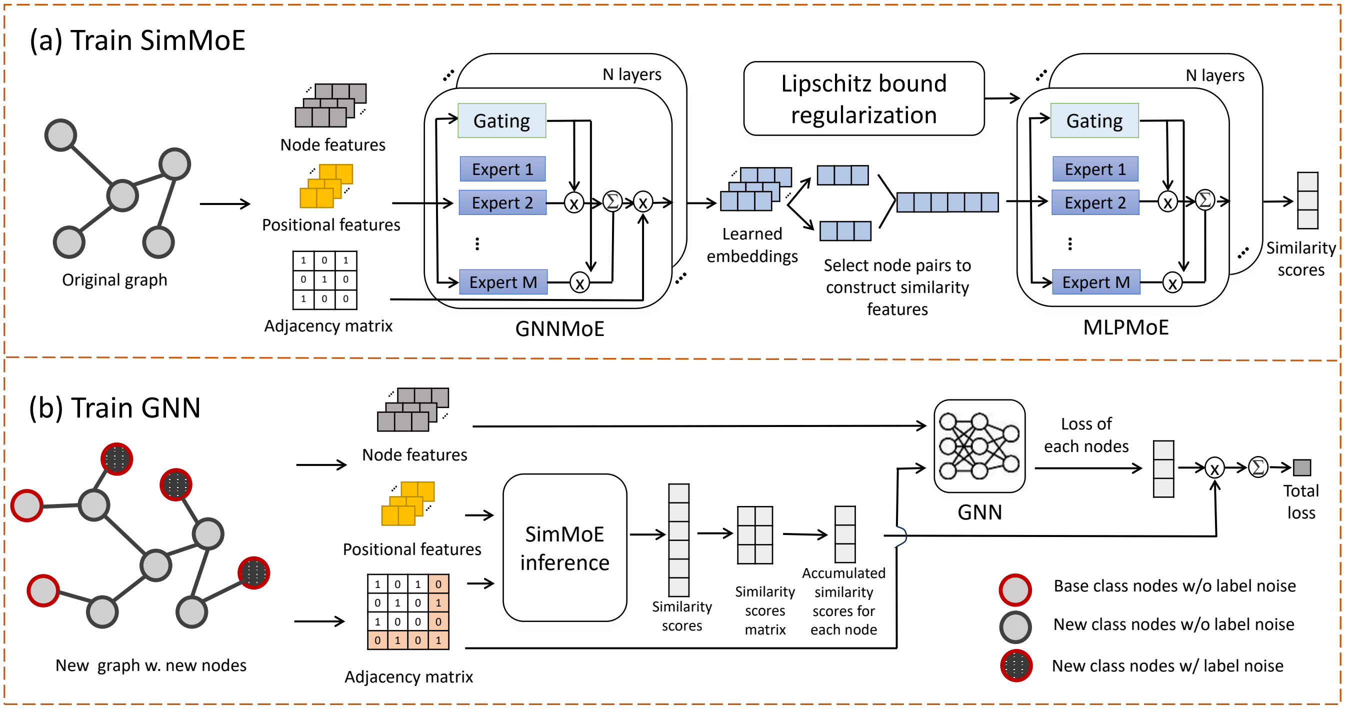

Figure 2: Our framework. (a) First, SimMoE which contains GNNMoE and MLPMoE is trained on the original graph. SimMoE is trained to predict whether two nodes are similar or not in terms of their labels. The pipeline of SimMoE can be divided into three steps: concatenating position embedding, encoding node features by GNNMoE, and predicting similarity scores by MLPMoE. (b) Second, GNN is trained for classification with the support of weighted loss. We utilize the trained SimMoE to infer the graph injected by new nodes and calculate accumulated similarity scores for each node, where lower scores indicate wrong labels. Finally, the similarity scores are taken as the weight of node losses to enforce model discard knowledge from nodes with wrong labels during the training process.

$p_i(h_v)$, which can be obtained using a softmax function:

$$p_i(h) = \frac{\exp(t(h)_i + \varepsilon_i)}{\sum_{k=1}^{N_E} \exp(t(h)_k + \varepsilon_k)}, \tag{7}$$

where $t(\cdot)$ is a linear transformation, and N_E is the number of all experts. The activated expert indices in the LinMoE module are determined by a gate-value $p_i(h)$. To illustrate, $p_i(h)$ takes the logits of all experts, obtained through a linear transformation of the input feature vector h, and computes the activation probability of each expert. The logits are weighted by the i-th value $t(h)_i$ of the linear transformation, and a random noise term ε_i is added to ensure randomness in the expert activating procedure. ε_i is typically chosen to be a sample from a Gaussian distribution. Each layer of the GNNMoE encoder first transforms the features of the target node and its neighboring nodes using the $\text{LinMoE}^{(l)}(\cdot)$ and then aggregates the transformed features of the neighboring nodes using $\text{AGG}(\cdot)$, which can be formulated as follows:

$$\begin{aligned} h_v^{(l)} = &\text{COMB}^{(l)}\Big(\text{LinMoE}^{(l)}(h_v^{(l-1)}), \\ &\text{AGG}\Big(\Big\{\text{LinMoE}^{(l)}(h_u^{(l-1)}), \forall u \in N_v\Big\}\Big)\Big), \end{aligned} \tag{8}$$

where $\text{AGG}(\cdot)$ and $\text{COMB}(\cdot)$ represent the neighbor aggregation and combination functions, respectively. N_v denotes the set of neighboring nodes of node v, and $h_v^{(l-1)}$ is the node representation at the l-th layer.

Predicting Similarity Scores by MLPMoE. After obtaining node embeddings $\boldsymbol{Z}$ (the output of the last layer of GNNMoE), we pair node embeddings to form node similarity features $\boldsymbol{E}$. To decrease the complexity, $\boldsymbol{E}$ are constructed by randomly concatenating two node embeddings, which is a random sampling process to sample N_t node pairs from total N^2 node pairs. We also discuss different sampling strategies in Appendix **??**. MLPMoE takes $\boldsymbol{E}$ as the input to output similarity scores for each node pair. Same with GNNMoE, we need to improve the generalization of the similarity predictor when we employ the SimMoE to infer the new graph. Therefore, we incorporate LinMoE to formulate MLPMoE, which updates the node pair similarity representation $\boldsymbol{E}^{(l)}$ in layer l as follows:

$$\boldsymbol{E}^{(l)} = \text{LinMoE}(\boldsymbol{E}^{(l-1)}). \tag{9}$$

Although the MLPMoE improves the generalization, as the training process iterates, the model still gradually overfits the training data while losing generalization. To solve this problem, we further introduce the Lipschitz bound to gain better generalization by stabilizing the output of the MLPMoE when the distribution shift happens during model inference. Here we first analyze MLPMoE and give its theoretical Lipschitz bound, then utilize it to regularize the training process. Specifically, consider MLPMoE to be represented as $f : \boldsymbol{E} \in \mathbb{R}^{N_t \times F^{\text{in}}} \rightarrow \boldsymbol{Y} \in \mathbb{R}^{N_t \times F^{\text{out}}}$, where $\boldsymbol{E}$ is the input feature matrix, $\boldsymbol{Y}$ is the output matrix, and N_t is the number of similarity features. To analyze the stability of the model output, we examine the Lipschitz bound of the Jacobian matrix of MLPMoE by introducing the following lemma and proposition. The detailed proofs are provided in Appendix A.

Lemma 4.1. *Let g be a Lipschitz continuous function. Denote g_i to be the i-th layer of g, $i = 1, \cdots, m$. Then the Lipschitz constant of function g satisfies:*

$$\operatorname{Lip}(g) \leq \left\| [\operatorname{Lip}(g_i)]_{i=1}^m \right\|, \tag{10}$$

where $[\operatorname{Lip}(g_i)]_{i=1}^m$ denotes the m-dimensional vector whose i-th component is $\operatorname{Lip}(g_i)$.

Lemma 4.1 establishes the connection between the Lipschitz constant of the entire model and those of its constituent subnetworks in a cascaded arrangement. Based on Lemma 4.1, we present the following proposition:

Proposition 4.2. *Let $\boldsymbol{Y}$ be the output of an L-layer MLPMoE (represented in $f(\cdot)$) with $\mathbf{E}$ as the input. Assuming the activation function (represented in $\rho(\cdot)$) is ReLU with a Lipschitz constant of $\operatorname{Lip}(\rho) = 1$, then the global Lipschitz constant of the Mixture-of-Experts network, denoted as $\operatorname{Lip}(f)$, satisfies the following inequality:*

$$\operatorname{Lip}(f) \leqslant \max_j \prod_{l=1}^{L} \left\| F^l \right\| \left\| \left[\textstyle\sum_{k=1}^{K_l} p_l^k \mathcal{J}^k(h^l) \right]_j \right\|_\infty, \tag{11}$$

where F^l represents the output dimension of the l-th Mixture-of-Experts layer which contains K_l experts; j is the index of the node; k is the index of the expert in the l-th layer; p_l^k is the gate value for k-th expert in the l-th layer and the vector $\left[\mathcal{J}^k(h^l)\right]$ is as follows:

$$\left[\mathcal{J}^k(h^l)\right] = \left[\left\| J_1^k(h^l) \right\|, \left\| J_2^k(h^l) \right\|, \cdots, \left\| J_{F^l}^k(h^l) \right\| \right]. \tag{12}$$

Notably, $J_i^k(h^l)$ denotes the input and output of the i-th row of the Jacobian matrix of the k-th expert in the l-th layer.

Proposition 4.2 provides a method to estimate the Lipschitz constants $\operatorname{Lip}(f)$ of similarity network $f(\cdot)$ by estimating Lipschitz constants of subnetworks. However, it is challenging to consider the potential cumulative effect in the similarity network. Therefore, we consider MLPMoE as a unitary model and directly derive the Lipschitz bound from the input and output of MLPMoE. To achieve this, we define the Jacobian matrix of i-th nodes pair similarity feature as $[J_i]_{F^{\text{out}} \times F^{\text{in}}} = \left[J_{i1}^\top, J_{i2}^\top, \cdots, J_{iF^{\text{out}}}^\top \right]^\top$, where $J_{ij} = \left[\frac{\partial Y_{ij}}{\partial E_{i1}}, \frac{\partial Y_{ij}}{\partial E_{i2}}, \cdots, \frac{\partial Y_{ij}}{\partial E_{iF^{\text{in}}}} \right]^\top$. Then we define $\mathcal{J}$ as follows:

$$\mathcal{J} = \left[\mathcal{J}_1^\top, \mathcal{J}_2^\top, \cdots, \mathcal{J}_N^\top \right]^\top, \tag{13}$$

where $\mathcal{J}_i = [\|J_{i1}\|, \|J_{i2}\|, \cdots, \|J_{iF^{\text{out}}}\|]^\top$. According to the definition of $\mathcal{J}$, we take the l_2*-norm* for each row of $\mathcal{J}$ and then take the infinite norm for the entire $\mathcal{J}$ to obtain the Lipschitz bound of the whole MLPMoE, which is formulated as follows:

$$\operatorname{Lip}(f) = \|\mathcal{J}\|_{\infty,2}. \tag{14}$$

During the training, the Lipschitz bound for the model output is computed using the gradients and norms of the input node pair features. We utilize this Lipschitz bound as a regularization term in the loss function, ensuring that the model's output stays within the defined constraints when we generalize the MLPMoE for inference. Equation (14) provides a strict Lipschitz bound from a theoretical perspective, and in our empirical implementation, we multiply a factor $\frac{Lip(E_{min})}{\frac{1}{N}\sum_1^N Lip(E_i)}$ to this Lipschitz bound to "tighten" it, where $Lip(E_{min})$ denotes the minimal expert's Lipschitz bound, and $Lip(E_i)$ denotes Lipschitz bound for i-th expert. In MoE, experts are different from each other because of different initialization and dispatched nodes, but the overall distributions of dispatched nodes to each expert are similar. Therefore, in this factor, we greedily chase the possible smallest Lipschitz bound ($Lip(E_{min})$ part), while tracing the overall Lipschitz bound ($\frac{1}{N}\sum_1^N Lip(E_i)$ part) of MLPMoE. The final loss for training SimMoE can be formulated as:

$$\mathcal{L}^{sim} = \mathcal{L}^s + \frac{Lip(E_{min})}{\frac{1}{N}\sum_1^N Lip(E_i)} \operatorname{Lip}(f), \tag{15}$$

where $\mathcal{L}^s$ is the training loss supervised by labels for similarity features (The label of the similarity feature is similar/dissimilar if two nodes have the same/different labels).

4.2 Training GNN on the New Graph with Weighted Node Classification Loss

After SimMoE is trained, we employ the SimMoE to infer the graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}', \mathcal{X}')$ which is associated with new nodes to obtain similarity scores. Note that during inference, the process of forming similarity features is different from training. Specifically, for nodes $\mathcal{V}'$ with $|C|$ class, we randomly sample N_i nodes from each class to form the nodes set $\mathcal{V}_s'$, where $|\mathcal{V}_s'| = N_i|C|$. Then, we pair each node from $\mathcal{V}'$ to each node from $\mathcal{V}_s'$ to construct similarity features. Finally, we take the output of similarity scores and construct a similarity scores matrix $\boldsymbol{S} \in \mathbb{R}^{N \times N_i}$. The element $\boldsymbol{S}_{i,j}$ represents the similarity score between the node $v' \in \mathcal{V}'$ and node $v_s' \in \mathcal{V}_s'$ that has the same label with v_i'. Finally, we calculate the accumulated similarity score $w_{v'} = \frac{1}{c}\sum_{j=1}^{c} S_{v',v_s'}$ for the node v', where $w_{v'}$ is the weight for downstream GNN classification training loss for node v' and j represents j-th sampled node with the same labels as node v'. Formally, this process can be formulated as follows:

$$\mathcal{L} = \frac{1}{|\mathcal{V}'|} \sum_{v' \in \mathcal{V}'} -w_i y_{v'}' \log[f(x_{v'})], \tag{16}$$

where $y_{v'}'$ is the label for node v'. Since we calculate the accumulated similarity scores between the nodes with the same label, the outlier and minority nodes that do not belong to their labeled class have lower accumulated similarity scores (weights). Lower weights indicate the wrong annotations, and accordingly, these nodes contribute less to learning the GNN classifier. This helps GNN avoid being fooled by noise labels during the training process.

Table 1: Node classification accuracy performance comparison of different methods. N.R. denotes label noise rate on novel classes and w.o. represents no label noise. The best result is bolded and the runner-up is underlined.

Dataset	N.R.	GCN [16]	CP-GNN [54]	NRGNN [4]	RTGNN [27]	GPPT [30]	JacGCN [43]	SimTrans [2]	LIME
Cora	w.o.	86.83 ± 0.38	86.72 ± 0.31	87.64 ± 0.40	85.61 ± 0.31	86.72 ± 0.33	85.20 ± 0.38	86.79 ± 0.22	**88.34 ± 0.32**
	0.1	82.47 ± 0.55	83.47 ± 0.72	84.13 ± 0.48	84.94 ± 0.62	83.32 ± 0.36	83.32 ± 0.45	84.24 ± 0.43	**86.94 ± 0.24**
	0.3	77.86 ± 0.35	78.04 ± 0.69	79.23 ± 0.48	82.95 ± 0.57	81.70 ± 0.51	79.00 ± 0.38	81.85 ± 0.40	**85.37 ± 0.52**
CiteSeer	w.o.	73.80 ± 0.32	70.92 ± 0.40	71.70 ± 0.15	72.66 ± 0.65	73.56 ± 0.42	72.75 ± 0.36	72.54 ± 0.28	**75.82 ± 0.33**
	0.1	67.58 ± 0.76	70.68 ± 0.71	71.40 ± 0.26	72.00 ± 0.33	69.59 ± 0.56	67.73 ± 0.61	69.92 ± 0.32	**75.28 ± 0.70**
	0.3	62.98 ± 0.57	64.66 ± 0.39	63.64 ± 0.64	67.91 ± 0.57	66.38 ± 0.61	65.83 ± 0.77	67.43 ± 1.10	**74.32 ± 0.32**
A-Photo	w.o.	**94.89 ± 0.08**	93.70 ± 0.88	93.91 ± 0.19	92.84 ± 0.19	92.54 ± 0.34	91.33 ± 0.46	93.07 ± 0.19	94.51 ± 0.18
	0.1	91.29 ± 1.36	92.82 ± 0.20	92.67 ± 1.06	90.95 ± 0.46	92.75 ± 0.26	92.25 ± 3.22	92.58 ± 0.77	**93.25 ± 0.24**
	0.3	87.99 ± 2.40	87.87 ± 4.77	92.60 ± 1.27	91.96 ± 0.74	91.03 ± 0.47	90.39 ± 3.53	90.48 ± 0.23	**93.79 ± 0.28**
A-Computer	w.o.	89.83 ± 3.33	88.95 ± 0.40	87.84 ± 1.08	88.20 ± 1.30	90.33 ± 0.62	88.21 ± 1.04	87.61 ± 0.71	**90.28 ± 1.08**
	0.1	85.91 ± 3.92	88.62 ± 2.14	86.51 ± 0.66	86.04 ± 3.26	88.00 ± 2.94	87.27 ± 2.36	85.91 ± 3.92	**89.45 ± 0.21**
	0.3	80.44 ± 7.31	79.13 ± 5.54	83.64 ± 1.04	83.46 ± 2.81	85.83 ± 6.37	84.92 ± 5.85	83.37 ± 5.00	**89.13 ± 1.22**
Actor	w.o.	26.09 ± 0.73	26.67 ± 0.51	26.87 ± 0.41	26.95 ± 0.63	26.95 ± 0.63	26.97 ± 0.34	27.33 ± 0.44	**29.17 ± 0.51**
	0.1	25.45 ± 0.46	26.37 ± 0.52	26.12 ± 0.31	25.25 ± 0.37	27.13 ± 0.30	27.12 ± 0.68	26.88 ± 0.14	**29.68 ± 0.34**
	0.3	25.22 ± 0.65	25.86 ± 0.26	25.70 ± 0.35	25.96 ± 0.46	26.75 ± 0.47	25.80 ± 0.77	26.71 ± 0.59	**29.68 ± 0.24**
Wisconsin	w.o.	47.20 ± 2.40	43.60 ± 1.96	40.40 ± 1.96	44.80 ± 2.04	44.40 ± 2.65	43.20 ± 2.04	46.40 ± 2.65	**48.40 ± 2.65**
	0.1	34.00 ± 1.26	37.60 ± 1.50	36.00 ± 2.19	38.80 ± 0.98	35.20 ± 1.60	36.40 ± 1.96	37.60 ± 1.50	**44.00 ± 2.83**
	0.3	27.60 ± 1.96	33.60 ± 1.96	27.20 ± 1.60	35.60 ± 3.67	35.60 ± 3.44	26.40 ± 4.08	31.20 ± 2.40	**38.80 ± 2.40**

5 EXPERIMENTS

This section presents comprehensive experiments to evaluate the effectiveness of our method when training on nodes with novel and noise labels. The experiments aim to address the following key research questions: **RQ-1**: Can our method outperform other state-of-the-art (SOTA) methods in terms of graph cross-supervised learning? **RQ-2**: What is the contribution of each component w.r.t. robustness against noise labels? **RQ-3**: How does our method generalize similarity when data is out-of-distribution? **RQ-4**: How do similarity scores enhance representations learning on the graph?

5.1 Setup

Datasets. We use six widely used PyG benchmark datasets with different graph types, i.e., Cora, CiteSeer, A-Computers, A-Photo, Actor, and Wisconsin, to evaluate our method. In the experiments, we randomly split datasets into 80% for training, 10% for validation, and 10% for testing. To evaluate the performance of LIME on large graphs, we use Flickr, Reddit, and AMiner datasets with default splits from GRB [57] benchmark datasets. The statistics of datasets used in the experiments are listed in Appendix B.

Baselines. We compare our method with five types of baselines. First, we compare the general GNN model GCN [16]. Second, we compare popular graph weakly supervised learning methods, including CP-GNN [54], NRGNN [4], RTGNN [27]. Third, we consider the graph prompt learning method GPPT [30]. Fourth, we compare popularly graph few-shot learning methods GLITTER [36], TENT [37], and COSMIC [38]. Note that graph few-shot learning methods have worse performances and scalability issues compared with other baselines, we present their results in Appendix B.4. Finally, we adapt the cross-supervised learning method SimTrans [2] for computer vision tasks to graphs. We also adapt Jaccard-GCN [43] to our setting by calculating Jaccard similarity scores to replace end-to-end similarity calculation in LIME.

Evaluation Setting. To adapt these datasets to our graph cross-supervised learning setting, we construct two graphs from each original graph by categories. We split the original category set C to half ($\frac{\lceil |C| \rceil}{2}$) base category set C_b and half ($\frac{\lfloor |C| \rfloor}{2}$) novel category set C_n, where $C_b \cup C_n = C$ and $C_b \cap C_n = \emptyset$. After that, we construct the graph with label noise rate $r \in \{0, 0.1, 0.3\}$ for training. The rate r is formally defined as the probability of a new training node v having another label among novel categories. Finally, we train our method and baselines on the constructed graph with label noises. We evaluate node classification accuracy according to the test set with the ground-truth labels.

Implementation. The implementation details of the LIME model in PyTorch-style pseudocode are displayed in Appendix B.2.

5.2 Performance Comparison

To answer the first research question **RQ-1**, we evaluate the graph cross-supervised performance of LIME and compare it with other SOTA baselines. The results are reported in Table 1. According to the table, we can observe that LIME comprehensively outperforms other baselines under 3 different label noise rates (w.o. noise, noise rate 0.1, and noise rate 0.3) across six graph datasets. Specifically, when nodes with novel class have label noise, LIME demonstrates impressive robust accuracy under two different label noise rates across all the datasets, while other baseline methods experience a severe decrease in accuracy as the label noise rate increases. For instance, on the Cora and CiteSeer datasets, LIME outperforms the most competitive baseline RTGNN by 2.00% and 3.28% when the label noise rate is 0.1, respectively. LIME outperforms RTGNN by

Table 2: Ablation studies for our method on three graph datasets of varying label noise rate. N.R. denotes label noise rate on novel classes and w.o. represents no label noise. The best result is bolded and the runner-up is underlined.

Dataset	Cora			CiteSeer			A-Photo		
N.R.	w.o.	0.1	0.3	w.o.	0.1	0.3	w.o.	0.1	0.3
GCN	86.83 ± 0.38	82.47 ± 0.55	77.86 ± 0.35	73.80 ± 0.32	67.58 ± 0.76	62.98 ± 0.57	94.89 ± 0.08	91.29 ± 1.36	87.99 ± 2.40
w.o. Position	87.79 ± 0.14	86.72 ± 0.35	85.12 ± 0.72	74.44 ± 0.59	74.59 ± 0.53	70.14 ± 0.43	94.59 ± 0.13	92.96 ± 0.76	93.75 ± 2.40
w.o. MLPMoE	87.38 ± 0.25	86.53 ± 0.26	84.58 ± 0.36	76.33 ± 0.40	73.62 ± 0.28	73.17 ± 0.36	94.59 ± 0.77	92.45 ± 0.50	91.65 ± 0.80
w.o. GNNMoE	87.12 ± 0.35	86.63 ± 0.13	84.71 ± 0.39	**76.93 ± 0.76**	74.22 ± 0.56	73.07 ± 0.27	**94.89 ± 0.44**	92.68 ± 0.32	91.88 ± 1.07
w.o. Lipshitz	88.34 ± 0.27	84.54 ± 0.49	78.81 ± 0.68	75.64 ± 0.55	74.53 ± 0.42	64.72 ± 0.92	93.19 ± 0.60	91.47 ± 0.36	89.43 ± 0.80
LIME	**88.34 ± 0.32**	**86.94 ± 0.24**	**85.37 ± 0.52**	75.82 ± 0.33	**75.28 ± 0.70**	**74.32 ± 0.32**	94.51 ± 0.18	**93.25 ± 0.24**	**93.79 ± 0.28**

Table 3: The performances of LIME and scalable baselines on three large-scale graphs. N.R. denotes label noise rate and w.o. represents no label noise. The best result is bolded.

Dataset	N.R.	GCN	GPPT	SimTrans	LIME
Flickr	w.o.	52.19±0.45	53.34±0.33	51.65±0.26	**53.96±0.20**
	0.1	48.57±0.90	51.03±1.01	50.26±1.57	**52.82±0.78**
	0.3	41.98±1.25	43.27±0.56	45.98±0.48	**51.39±2.01**
Reddit	w.o.	93.35±0.24	**93.45±0.19**	92.89±0.13	93.30±0.40
	0.1	90.33±0.20	90.86±0.21	90.12±0.34	**92.03±0.13**
	0.3	80.76±1.57	83.03±0.75	83.87±53	**89.74±2.38**
AMiner	w.o.	63.41±0.23	**65.03±0.55**	64.33±0.12	64.46±0.23
	0.1	58.36±0.21	61.35±0.24	62.35±0.66	**63.55±0.78**
	0.3	50.30±0.32	52.54±1.18	56.40±2.52	**62.45±0.95**

2.42% on Cora and 6.41% on CiteSeer when the label noise rate increases to 0.3. On the Amazon dataset, compared to the most competitive baselines, LIME outperforms the runner-up baselines by 3.53% when the label noise rate is 0.1 and by 0.43% when the label noise rate increases to 0.3 on the Photo dataset. On the Computer dataset, our method outperforms the runner-up baselines by 0.83% when the label noise rate is 0.1 and by 3.30% when the label noise rate increases to 0.3. On the Actor dataset, LIME outperforms the most competitive baseline GPPT by 2.55% when the label noise rate is 0.1 and by 2.93% when the label noise rate increases 0.3. On the Wisconsin dataset, our method outperforms the runner-up baseline RTGNN by 5.20% when the label noise rate is 0.1 and by 3.30% when the label noise rate increases 0.3. In addition, our LIME also demonstrates strong representation ability on clean graphs without label noise and outperforms other baselines. This observation indicates that our model discriminates many outlier nodes and assigns them lower loss weight during training, which helps the model concentrate on learning representations from nodes with better quality. In summary, the above results showcase the effectiveness of LIME against label noise against different noise rates on the graph datasets, facilitated by our Lipschitz bounded Mixture-of-Experts similarity network.

5.3 Scale to Large Graphs

In the realm of graph learning, handling large-scale graphs efficiently remains a formidable challenge due to the complexity and size of real-world networks. While Section 1 reviews several graph learning baselines designed to mitigate label noise, prominent methods such as CP-GNN [54], NRGNN [4], and RTGNN [27] encounter scalability issues, primarily due to Out-of-Memory (OOM) problems. This limitation underscores the need for scalable solutions capable of accommodating the vast node sets characteristic of large graphs. Fortunately, Our LIME method has time and memory linear complexities and is able to scale to large graphs. Let’s denote N as the number of nodes on a graph, After removing the other constants, the time and memory complexities for training and inference are both $O(N)$, which is **linear** to the number of nodes N. The full analysis and computational overhead can be found in Appendix C. The linear complexity predicts the scalability of LIME, and the practical scalability and efficacy of LIME are empirically validated through comparative experiments on three large-scale graph datasets. As reported in Table 3, LIME not only demonstrates superior scalability but also maintains robust performance in scenarios with prevalent label noise, thereby affirming its potential as a scalable and effective solution for graph cross-supervised problems.

5.4 Ablation Study

LIME integrates the positional embedding, the network with Mixture-of-Experts, and the corresponding Lipshitz bound to improve the generalization of similarity network. Thus, to answer the second research question **RQ-2**, we remove each of these components and conduct ablation experiments. The results are shown in Table 2. In particular, we ablate the model as (a) without positional embedding (w.o. Position.), (b) without Mixture-of-Experts in MLPMoE (w.o. MLPMoE), (c) without Mixture-of-Experts in GNNMoE (w.o. GNNMoE), (d) without the Lipshitz bound (w.o. Lipshitz), and (e) using the vanilla GCN instead of LIME. The results show that removing any component of the LIME decreases its performance when novel classes contain noises. This highlights the critical role each component plays in generalizing the similarity knowledge and its ability to provide robustness against label noises. For (a) **w.o. Struc.**, removing the positional embedding of the similarity network decreases LIME’s ability to learn position similarity knowledge. For example, on the CiteSeer dataset, the model experiences a 0.69% loss in accuracy when the label noise rate is 0.1, and a 4.18% loss in accuracy when the label noise rate increases to 0.3. For (b) **w.o. MLPGNN**, removing the MoE in MLPMoE prevents the model from learning diverse similarity representations, indicating lower model capacity limits the model’s generalization to unseen data distribution. On the Cora dataset, the removal of this component results in a 0.41% loss in accuracy when the label noise rate is 0.1, and a 0.79% loss in accuracy when the noise rate increases to 0.3. For (c) **w.o. GNNMoE**, removing MoE in GNNMoE prevents the

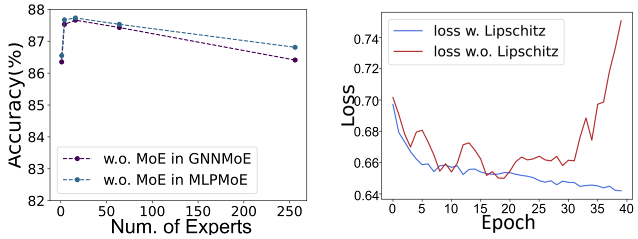

Figure 3: The classification accuracy with different numbers of experts (left) and the inference loss on novel classes when training on base classes on the Cora dataset (right).

model from learning diverse node representations. On the Photo dataset, the removal of this component results in a 0.57% loss in accuracy when the label noise rate is 0.1, and a 1.81% loss in accuracy when the noise rate increases to 0.3. For (d) **w.o. Lipshitz**, removing the Lipshitz bound for the Mixture-of-Experts network decreases the model's performance due to the loss of generalization for similarity. On the Cora dataset, it results in a 2.40% decrease in accuracy when the noise rate is 0.1, and a 6.56% loss in accuracy when the noise rate decreases to 0.3. For (e) **Vanilla GCN**, removing all components degenerates LIME into a vanilla GCN model. This results in a 7.51% decrease in accuracy on the Cora dataset, an 11.34% loss in accuracy on the CiteSeer dataset, and an 8.69% loss on the Computer dataset when the label noise rate is 0.3. This ablation study demonstrates the contribution of each component in the proposed LIME model and shows the effectiveness of LIME in improving the GNNs' performance against noise labels.

5.5 What Wins Better Performance Against Label Noise?

To address the third research question **RQ-3**, we investigate the main generalizing contribution of our model provided by the Mixture-of-Experts as well as corresponding Lipshitz bounds and analyze how they generalize similarity knowledge.

Mixture-of-Experts. To generalize the model to node pair features with distribution shift, we introduce the Mixture-of-Experts to the similarity network. SimMoE model contains multiple experts, thus it obtains diverse representations and improves the generalization. As shown in Figure 3, models with more experts bring better performance of LIME. Although too many experts decrease the model performance because each expert is dispatched with too little data, i.e., the data-hungry problem, it can be solved by sampling more nodes to form more similar features in SimMoE during training.

Lipshitz Bound. To study the effectiveness of Lipshitz bound during generalizing similarity inference when a data distribution shift happens, we plot the loss curve in Figure 3. The results show that as the training epoch increases, thanks to the generalization ability learned by the Lipshitz bounded network, the loss of the similarity network equipped with LIME continues to decrease. On the other hand, the similarity network without the Lipshitz bound gradually overfits the training data and loses generalization. This demonstrates how the Lipshitz bound generalizes similarity knowledge.

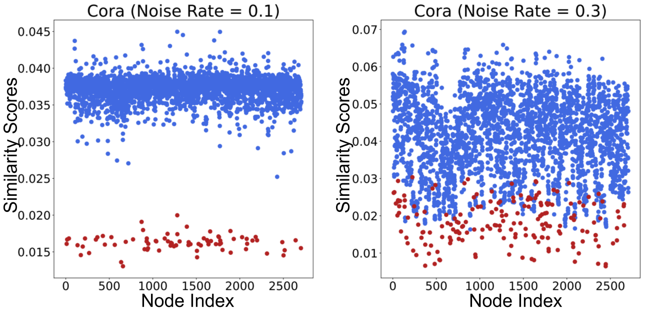

Figure 4: The similarity scores of nodes with indexes on the Cora dataset. Blue nodes denote nodes without label noise and red nodes denote nodes with label noise. Nodes with label noise will be recognized and separated from nodes without label noise in terms of similarity scores by LIME. The smaller similarity scores indicate label noise. The higher label noise rate makes it harder to recognize nodes with label noise.

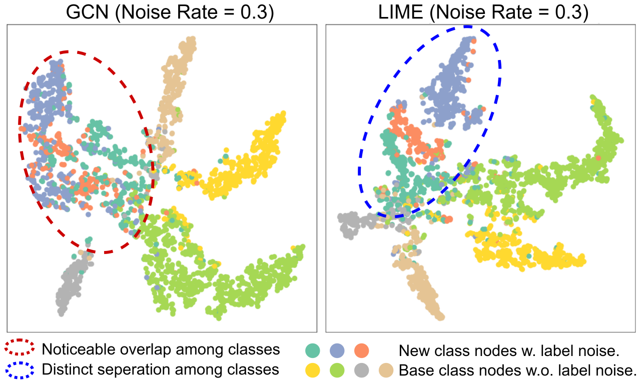

Figure 5: T-SNE visualization depicting the node representations obtained by vanilla GCN and LIME on the Cora dataset. GCN is fooled by label noise and wrongly clusters node representations even if these nodes have different ground-truth labels (in the red dashed circle). However, our LIME model still performs well on the clustering of these nodes with label noise (in the blue dashed circle).

5.6 How Similarity Scores Enhance Cross-Supervised Learning against Noise

To address the fourth research question **RQ-4**, we explore how obtained similarity scores enhance the graph representation learning when the noise is assigned to the labels. As Figure 4 shows, the similarity network outputs the scores for each node. Generally, the similarity scores for nodes without label noise (blue) are higher than the scores for nodes with label noise (red). When the label noise rate is low, i.e. 0.1, nodes with label noise will be easily recognized and separated from nodes without label noise. When the label noise rate increases to 0.3, the scores for nodes with and without label noise are relatively closer, which indicates that more label noise hinders the right calculation of similarity scores. However, we observe that the average similarity scores for nodes with label noise are still lower than that for nodes without label noise. After obtaining the similarity scores, we take them as the weight of each node's loss to minimize the negative effect of nodes with label noise.

5.7 Embedding Visualization

Noise labels mistakenly guide the model to map node embeddings close even if they have different ground-truth labels. As Figure 5 shows, vanilla GCN is fooled by label noise. It wrongly clusters node representations even if these nodes have different true labels. This phenomenon becomes even worse when the label noise rate increases from 0.1 to 0.3. In contrast, LIME performs well on the clustering of these nodes with label noises. Moreover, LIME also maps some node representations to the cluster (class) that has a different label and no label noises, while having better performance under w.o. noise setting in Table 1. We explain this phenomenon as LIME can assign lower loss weight to some outlier or wrongly labeled training nodes and learn better representations.

6 CONCLUSION

In this paper, we are the first to formulate one practical problem, *Graph Cross-Supervised (Weak-Shot) Learning*, from the real world, which describes the need to model new nodes with novel classes and potential label noises. To solve this problem, we propose LIME, a novel model to encode new nodes and handle label noises. In particular, LIME trains the similarity network to capture the knowledge from the original classes, aiming to obtain insights for the emerging novel classes. We also employ the MoE techniques and Lipschitz bound to increase the generalization of the similarity network. By utilizing LIME to calculate loss weight for nodes with potential label noise, the GNN classifier avoids the negative effect of new nodes with noise labels. Our experimental results show the effectiveness of LIME in encoding new nodes with novel categories and the superiority in handling label noises for graph cross-supervised learning problems and real-world applications.

7 ACKNOWLEDGMENTS

Thanks to Prof. Yang Liu at UC Santa Cruz, who supported this work by providing many precious insights. Thanks to Yifei Wang at Brandeis University for supporting intuitive figures in the experiment section. This work was partially supported by Dartmouth, the NSF under grants IIS-2321504, IIS-2334193, IIS2203262, IIS-2217239, CNS-2203261, IIS-2340346, and CMMI-2146076. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

REFERENCES

[1] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[2] Junjie Chen, Li Niu, Liu Liu, and Liqing Zhang. Weak-shot fine-grained classification via similarity transfer. In *NeurIPS*, 2021.

[3] Ming Cheng, Ziyi Zhou, Bowen Zhang, Ziyu Wang, Jiaqi Gan, Ziang Ren, Weiqi Feng, Yi Lyu, Hefan Zhang, and Xingjian Diao. Efflex: Efficient and flexible pipeline for spatio-temporal trajectory graph modeling and representation learning. *arXiv preprint arXiv:2404.12400*, 2024.

[4] Enyan Dai, Charu Aggarwal, and Suhang Wang. Nrgnn: Learning a label noise resistant graph neural network on sparsely and noisily labeled graphs. In *KDD*, 2021.

[5] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *WWW*, 2019.

[6] Zichu Fei, Qi Zhang, and Yaqian Zhou. Iterative gnn-based decoder for question generation. In *EMNLP*, 2021.

[7] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. Large-scale learnable graph convolutional networks. In *KDD*, 2018.

[8] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, 2016.

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

[10] Judy Hoffman, Sergio Guadarrama, Eric S Tzeng, Ronghang Hu, Jeff Donahue, Ross Girshick, Trevor Darrell, and Kate Saenko. Lsda: Large scale detection through adaptation. In *NeurIPS*, 2014.

[11] Ronghang Hu, Piotr Dollár, Kaiming He, Trevor Darrell, and Ross Girshick. Learning to segment every thing. In *CVPR*, 2018.

[12] Tinglin Huang, Yuxiao Dong, Ming Ding, Zhen Yang, Wenzheng Feng, Xinyu Wang, and Jie Tang. Mixgcf: An improved training method for graph neural network-based recommender systems. In *KDD*, 2021.

[13] Yaning Jia, Chunhui Zhang, and Soroush Vosoughi. Aligning relational learning with lipschitz fairness. In *ICLR*, 2024.

[14] Hyunjik Kim, George Papamakarios, and Andriy Mnih. The lipschitz constant of self-attention. In *International Conference on Machine Learning*, 2021.

[15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

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

[17] Weicheng Kuo, Anelia Angelova, Jitendra Malik, and Tsung-Yi Lin. Shapemask: Learning to segment novel objects by refining shape priors. In *ICCV*, 2019.

[18] Guohao Li, Matthias Müller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *ICCV*, 2019.

[19] Guohao Li, Matthias Müller, Guocheng Qian, Itzel Carolina Delgadillo Perez, Abdulellah Abualshour, Ali Kassem Thabet, and Bernard Ghanem. Deepgcns: Making gcns go as deep as cnns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[20] Guohao Li, Matthias Müller, Bernard Ghanem, and Vladlen Koltun. Training graph neural networks with 1000 layers. In *ICML*, 2021.

[21] Yan Liu, Zhijie Zhang, Li Niu, Junjie Chen, and Liqing Zhang. Mixed supervised object detection by transferring mask prior and semantic similarity. In *NeurIPS*, 2021.

[22] Zheyuan Liu, Chunhui Zhang, Yijun Tian, Erchi Zhang, Chao Huang, Yanfang Ye, and Chuxu Zhang. Fair graph representation learning via diverse mixture-of-experts. WWW, 2023.

[23] Seth A Myers, Aneesh Sharma, Pankaj Gupta, and Jimmy Lin. Information network or social network? the structure of the twitter follow graph. In *WWW*, 2014.

[24] Hoang NT, Choong Jun Jin, and Tsuyoshi Murata. Learning graph neural networks with noisy labels. *arXiv preprint arXiv:1905.01591*, 2019.

[25] Zhongyu Ouyang, Chunhui Zhang, Shifu Hou, Chuxu Zhang, and Yanfang Ye. How to improve representation alignment and uniformity in graph-based collaborative filtering? In *International AAAI Conference on Web and Social Media*, 2024.

[26] Giorgio Patrini, Alessandro Rozza, Aditya Krishna Menon, Richard Nock, and Lizhen Qu. Making deep neural networks robust to label noise: A loss correction approach. In *CVPR*, 2017.

[27] Siyi Qian, Haochao Ying, Renjun Hu, Jingbo Zhou, Jintai Chen, Danny Z Chen, and Jian Wu. Robust training of graph neural networks via noise governance. In *WSDM*, 2023.

[28] Luana Ruiz, Fernando Gama, and Alejandro Ribeiro. Gated graph recurrent neural networks. *IEEE Transactions on Signal Processing*, 2020.

[29] Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *ICLR*, 2017.

[30] Mingchen Sun, Kaixiong Zhou, Xin He, Ying Wang, and Xin Wang. Gppt: Graph pre-training and prompt tuning to generalize graph neural networks. In *KDD*, 2022.

[31] Yijun Tian, Kaiwen Dong, Chunhui Zhang, Chuxu Zhang, and Nitesh V Chawla. Heterogeneous graph masked autoencoders. In *AAAI*, 2023.

[32] Yijun Tian, Chuxu Zhang, Zhichun Guo, Xiangliang Zhang, and Nitesh V Chawla. Learning mlps on graphs: A unified view of effectiveness, robustness, and efficiency. In *ICLR*, 2023.

[33] Yijun Tian, Huan Song, Zichen Wang, Haozhu Wang, Ziqing Hu, Fang Wang, Nitesh V Chawla, and Panpan Xu. Graph neural prompting with large language models. In *AAAI*, 2024.

[34] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[35] Aladin Virmaux and Kevin Scaman. Lipschitz regularity of deep neural networks: analysis and efficient estimation. In *NeurIPS*, 2018.

[36] Song Wang, Chen Chen, and Jundong Li. Graph few-shot learning with task-specific structures. *NeurIPS*, 2022.

[37] Song Wang, Kaize Ding, Chuxu Zhang, Chen Chen, and Jundong Li. Task-adaptive few-shot node classification. In *KDD*, 2022.

[38] Song Wang, Zhen Tan, Huan Liu, and Jundong Li. Contrastive meta-learning for few-shot node classification. In *KDD*, 2023.

[39] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. Kgat: Knowledge graph attention network for recommendation. In *KDD*, 2019.

[40] Yifei Wang, Shiyang Chen, Guobin Chen, Ethan Shurberg, Hang Liu, and Pengyu Hong. Motif-based graph representation learning with application to chemical molecules. In *Informatics*. MDPI, 2023.

[41] Qianlong Wen, Zhongyu Ouyang, Chunhui Zhang, Yiyue Qian, Chuxu Zhang, and Yanfang Ye. GCVR: Reconstruction from cross-view enable sufficient and robust graph contrastive learning. In *The 40th Conference on Uncertainty in Artificial Intelligence*, 2024.

[42] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *ICML*, 2019.

[43] Huijun Wu, Chen Wang, Yuriy Tyshetskiy, Andrew Docherty, Kai Lu, and Liming Zhu. Adversarial examples on graph data: Deep insights into attack and defense. In *IJCAI*, 2019.

[44] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[45] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*, 2018.

[46] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. In *NeurIPS*, 2020.

[47] Donghan Yu, Chenguang Zhu, Yiming Yang, and Michael Zeng. Jaket: Joint pre-training of knowledge graph and language understanding. In *AAAI*, 2022.

[48] Junliang Yu, Hongzhi Yin, Xin Xia, Tong Chen, Lizhen Cui, and Quoc Viet Hung Nguyen. Are graph augmentations necessary? simple graph contrastive learning for recommendation. In *SIGIR*, 2022.

[49] Xiangchi Yuan, Chunhui Zhang, Yijun Tian, and Chuxu Zhang. Navigating graph robust learning against all-intensity attacks. In *The Second Workshop on New Frontiers in Adversarial Machine Learning*, 2023.

[50] Xiangchi Yuan, Chunhui Zhang, Yijun Tian, Yanfang Ye, and Chuxu Zhang. Mitigating emergent robustness degradation while scaling graph learning. In *ICLR*, 2024.

[51] Chunhui Zhang, Chao Huang, Yijun Tian, Qianlong Wen, Zhongyu Ouyang, Youhuan Li, Yanfang Ye, and Chuxu Zhang. When sparsity meets contrastive models: Less graph data can bring better class-balanced representations. In *ICML*, 2023.

[52] Chunhui Zhang, Yijun Tian, Mingxuan Ju, Zheyuan Liu, Yanfang Ye, Nitesh Chawla, and Chuxu Zhang. Chasing all-round graph representation robustness: Model, training, and optimization. In *ICLR*, 2023.

[53] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V Chawla. Heterogeneous graph neural network. In *KDD*, 2019.

[54] Mengmei Zhang, Linmei Hu, Chuan Shi, and Xiao Wang. Adversarial label-flipping attack and defense for graph neural networks. In *2020 IEEE International Conference on Data Mining (ICDM)*, 2020.

[55] Yanfu Zhang, Shangqian Gao, Jian Pei, and Heng Huang. Improving social network embedding via new second-order continuous graph neural networks. In *KDD*, 2022.

[56] Da Zheng, Minjie Wang, Quan Gan, Zheng Zhang, and George Karypis. Learning graph neural networks with deep graph library. In *WWW*, 2020.

[57] Qinkai Zheng, Xu Zou, Yuxiao Dong, Yukuo Cen, Da Yin, Jiarong Xu, Yang Yang, and Jie Tang. Graph robustness benchmark: Benchmarking the adversarial robustness of graph machine learning. In *NeurIPS*, 2021.

[58] Yuanyi Zhong, Jianfeng Wang, Jian Peng, and Lei Zhang. Boosting weakly supervised object detection with progressive knowledge transfer. In *ECCV*, 2020.

A PROOFS

The proof of Lemma 4.1 is adapted from Lemma 1 in the research [13], aligning the notations and the scope of this paper.

A.1 Proof of Lemma 4.1 in Section 4

Proof. We observe that

$$\begin{aligned}\frac{\|g(\boldsymbol{x})-g(\boldsymbol{y})\|}{\|\boldsymbol{x}-\boldsymbol{y}\|} &= \frac{\left\|[g_i(\boldsymbol{x})-g_i(\boldsymbol{y})]_{i=1}^n\right\|}{\|\boldsymbol{x}-\boldsymbol{y}\|} \\ &= \left\|\left[\frac{|g_i(\boldsymbol{x})-g_i(\boldsymbol{y})|}{\|\boldsymbol{x}-\boldsymbol{y}\|}\right]_{i=1}^n\right\|.\end{aligned} \tag{17}$$

Furthermore, for each $i \in [n]$, $\frac{|g_i(\boldsymbol{x})-g_i(\boldsymbol{y})|}{\|\boldsymbol{x}-\boldsymbol{y}\|} \leq \mathrm{Lip}(g_i)$. Therefore, we can write

$$\begin{aligned}\mathrm{Lip}(g) &= \sup_{\boldsymbol{x}\neq\boldsymbol{y}} \frac{\|g(\boldsymbol{x})-g(\boldsymbol{y})\|}{\|\boldsymbol{x}-\boldsymbol{y}\|} \\ &= \sup_{\boldsymbol{x}\neq\boldsymbol{y}} \left\|\left[\frac{|g_i(\boldsymbol{x})-g_i(\boldsymbol{y})|}{\|\boldsymbol{x}-\boldsymbol{y}\|}\right]_{i=1}^n\right\| \\ &\leq \sup_{\boldsymbol{x}\neq\boldsymbol{y}} \left\|[\mathrm{Lip}(g_i)]_{i=1}^n\right\| = \left\|[\mathrm{Lip}(g_i)]_{i=1}^n\right\|,\end{aligned} \tag{18}$$

□

A.2 Proof of Proposition 4.2 in Section 4

Proof. Let Y denotes the output of an L-layer Mixture-of-Experts network with input $\boldsymbol{X}$. Assuming the commonly used ReLU activation function as the non-linear layer $\rho(\cdot)$, we have $\mathrm{Lip}(\rho) = 1$. First, we consider the Lipschitz constant between the hidden states of two node feature pairs output by any layer $h(\cdot)$ in $f(\cdot)$. Let $h(x_1)$ and $h(x_2)$ represent the hidden states of input feature x_1 and x_2, respectively. Since two inputs have the same activated experts, we assume each expert has a very close gate value, i.e. p_l^k, from a statistical perspective. By applying the triangle inequality, Lemma 4.1, we obtain:

$$\begin{aligned}\frac{\|h(x_1)-h(x_2)\|}{\|x_1-x_2\|} &= \frac{\left\|\sum_{k=1}^{K} p^k\,[h(x_1)-h(x_2)]\right\|}{\|x_1-x_2\|} \\ &\leqslant \left\|\frac{\sum_{i=1}^{F'}\sum_{k=1}^{m} p^k\left[h^k(x_1)_i - h^k(x_2)_i\right]}{\|x_1-x_2\|}\right\| \\ &\leqslant \left\|F' \times \max_i \frac{\sum_{k=1}^{m} p^k[h^k(x_1)_i - h^k(x_2)_i]}{\|x_1-x_2\|}\right\|,\end{aligned} \tag{19}$$

let $f(x)_j$ denotes the j-th column of the matrix $f(x)$. We denote $h_l^k(\cdot)$ as the k-th expert of the l-th layer in $f(\cdot)$, then by applying the conclusion from Lemma 4.1 and leveraging the Lipschitz property of the ReLU activation function, we have:

$$\frac{\left\|f(x_1)_{j,1}-f(x_2)_{j,2}\right\|}{\left\|x_{j,1}-x_{j,2}\right\|} \leqslant \prod_{l=1}^{L}\left\|F^{l'}\right\| \left\|\left[\sum_{k=1}^{K^l} p_l^k \mathcal{J}^k(h^l)\right]_j\right\|_\infty, \tag{20}$$

where $x_{j,1}$ and $x_{j,2}$ denote j-th nodes pair similarity features in x_1 and x_2, respectively, and $\left[\mathcal{J}(h^l)\right]_j$ represents the j-th node pair similarity feature's the Jacobian matrix of the l-th layer. Therefore, the Lipschitz constant for the MLPMoE can be expressed as:

$$\mathrm{Lip}(f) = \max_j \prod_{l=1}^{L}\left\|F^{l'}\right\| \left\|\left[\sum_{k=1}^{K^l} p_l^k p_l^k \mathcal{J}^k(h^l)\right]_j\right\|_\infty. \tag{21}$$

In summary, we have shown that for any two input samples x_1 and x_2, the Lipschitz constant of the MLPMoE, denoted as $\mathrm{Lip}(f)$, satisfies:

$$\|Y_1 - Y_2\| \leqslant \mathrm{Lip}(f)\,\|\boldsymbol{X}_1 - \boldsymbol{X}_2\|, \tag{22}$$

Algorithm 1: LIME: A simplified PyTorch-style Pseudocode of LIME.

```
for epoch in range(1, M):
    # Eq. (5), input: G=(A, X)
    X_P = Node2Vec(A), X_F = CONCAT(X_P,X)
    # Eq. (6) Encode node features
    Z_n = GNNMoE (A, X_F)
    # Pair node embeddings to obtain the similarity features
    Z = Pair(Z_n)
    # Eq.(9) Obtain similarity scores
    Scores_Prediction = MLPMoE(Z)
    # Eq. (14) Calculate the Lipschitz bound Lip(f)
    Lipschitz_bound = Lip(MLPMoE)
    # Eq. (15) Obtain final loss.
    Loss_sim = L_s + Lipschitz_bound
    Loss_sim.backward()
    optimizer.step()

# After new node arriving, G=(A, X) becomes to G'=(A', X')
# Utilizing trained SimMoE to infer the training nodes and
      obtain the similarity matrix S.
S = SimMoE_model(A',X')
# Sum each row of the similarity matrix to obtain the final
      similarity score W for each node.
W = Sum (S_i)
for epoch in range(1, N):
    # input: G'=(A', X')
    Cls_Prediction = GNN(A',X')
    # Eq. (16)Train GNN on downstream tasks with weighted loss.
    Loss = W*L_s
    Loss.backward()
    optimizer.step()
```

Table 4: Statistics of datasets used in the experiments.

Name	#nodes	#edges	#features	#classes
Cora	2,708	10,556	1,433	7
CiteSeer	3,327	9,104	3,703	6
Computers	13,752	491,722	767	10
Photo	7,650	238,162	745	8
Actor	7,600	30,019	932	5
Wisconsin	251	515	1,703	5
Flickr	89,250	449,878	500	7
Reddit	232,965	11,606,919	602	41
AMiner	659,574	2,878,577	100	18

where Y denotes the output of the MLPMoE for inputs X. This inequality implies that the Lipschitz constant $\mathrm{Lip}(f)$ controls the magnitude of changes in the output based on input distribution shift. Therefore, we have established the following result:

$$\|Y_1 - Y_2\| \leqslant \prod_{l=1}^{L} \left\|F^{l'}\right\| \left\| \left[\sum_{k=1}^{K^l} p_l^k \mathcal{J}^k(h^l) \right]_j \right\|_\infty \|X_1 - X_2\| . \qquad (23)$$

This inequality demonstrates that the Lipschitz constant of the MLPMoE, $\mathrm{Lip}(f)$, controls the magnitude of the difference in the output Y based on the difference in the input X. It allows us to analyze the stability of the model's output with respect to input distribution shift. □

B IMPLEMENTATION DETAILS

B.1 Implementation Details of LIME

We report the mean and standard deviation of ten independent runs with the same data splits and random seeds. We use two layers for the GCN encoder and the MLP in the similarity network. Four experts are utilized for each layer. In addition, we set the learning rate to 0.01 and the noisy gate rate to 0.01. The epoch number for training similarity network is 100 and for node classification is 500. We use Adam [15] to optimize the model. Both the SimMoE and the GNN classifier are implemented in PyTorch and trained on the NVIDIA V100 GPU. The hidden sizes for GNNMoE and MLPMoE are 32 and 16 respectively. For classification GCN, the hidden sizes are {16, 16, 64, 64, 32, 32, 64, 64, 64} for {Cora, CiteSeer, Photo, Computer, Actor, Wisconsin, Flickr, Reddit, AMiner} datasets. The statistics of different datasets are listed in Table 4.

B.2 Pseudo Code

Algorithm 1 displays a PyTorch-style pseudocode of LIME model in detail.

B.3 More Details about Baselines

Here we include more details about baselines in the experiment part. We keep the parameters and configurations of baselines the same as in the original paper.

- **GCN [16]:** Graph Convolution Network uses a localized first-order approximation of spectral graph convolutions as its convolutional architecture.
- **CP-GNN [54]:** This method proposes a defense framework that introduces a community-preserving self-supervised task as regularization to avoid overfitting on nodes with label noise.
- **NRGNN [4]:** NRGNN proposes to link the unlabeled nodes with labeled nodes of high feature similarity and utilize accurate pseudo labels to reduce the effects of label noise.
- **GPPT [30]:** For GPPT framework, pre-trained GNNs could be applied without tedious fine-tuning to evaluate the linking probability of token pair, and produce the node classification decision. We find this method has a strong performance when there exists label noise.
- **SimTrans [2]:** SimTrans transfers pairwise semantic similarity from base categories to novel categories with additional adversarial loss. SimTrans is originally designed for fine-grained image classification and we adapt it to graph data.
- **Jaccard-GCN [43]:** Jaccard-GCN is originally designed to empower the GCN model with strong robustness against graph structure perturbation. We adapt Jaccard-GCN to our setting by calculating Jaccard similarity scores to replace end-to-end similarity calculation in LIME.

B.4 More Baselines

We further experiment on 3 popular graph-based few-shot learning methods GLITTER [36], TENT [37], and COSMIC [38]. From experiment results in Table 5, we find graph-based few-shot learning methods perform better than Vallina GCN under the label noise. However, these methods are outperformed by the most recent graph weakly supervised baseline RTGNN and our method. This experimental observation aligns with previous computer vision research [2] on

Table 5: The performances of graph few-shot learning baselines and our LIME method on the Cora dataset.

Noise rate	GCN	RTGNN	GLITTER	TENT	COSMIC	LIME
0.0	86.83	85.61	84.03	85.54	86.96	**88.34**
0.1	82.47	84.94	83.21	82.85	84.84	**86.94**
0.3	77.86	82.95	80.34	80.32	82.47	**85.37**

cross-supervised learning, which shows that weakly supervised learning methods are more powerful in cross-supervised learning problems compared with few-shot learning methods.

C COMPLEXITY AND COMPUTATIONAL OVERHEAD

Complexity. Here we analyze the time and memory complexities of our method. Let N be the number of nodes, d be the size of the hidden channels (we assume it is of the same order as the size of the input features), l_1 be the number of encoder layers in the GNNMoE, l_2 be the number of MLPMoE layers, l_3 be the number of classification GNN layers, k be the number of experts for MoEs, n be sampled nodes from each class during constructing inference similarity matrix, our comprehensive complexity analysis is structured as follows:

Complexities for training. For each MoE layer in SimMoE, N/k nodes are distributed into each expert, resulting in $O(d^2N/k)$ time and $O(d^2 + dN/k)$ memory complexity. For all k experts, the total time complexity is $O(d^2N)$ and memory complexity is $O(kd^2 + dN)$. The gating network is trained with $O(dkN)$ time complexity and constant $O(dk)$ memory complexity for gating in one layer. For Lipschitz bound calculation, the time and memory complexities are $2dN$. For constructing similarity features, the time complexity is N and the memory complexity is $2dN$ Overall time complexity of SimMoE module is $O((l_1 + l_2)dN(d + k) + (2d + 1)N)$ and memory complexity is $O((l_1+l_2)d(N+kd+k)+4dN)$. For GNN classifier, the time complexity is $O(l_3d^2N)$ and the memory complexity $O(l_3dN+l_3d^2)$.

Complexities for inference. The complexity analysis of inference is similar to the training process but no Lipschitz bound calculation. For constructing the similarity matrix and obtaining similarity scores, the time complexity is $2nN$ and the memory complexity is $(2d + 1)nN$. Overall time complexity of SimMoE module is $O((l_1 + l_2n)dN(d+k)+2nN)$ and memory complexity is $O((l_1+l_2n)d(N + kd + k) + (2d + 1)nN)$. For GNN classifier, the time complexity is $O(l_3d^2N)$ and the memory complexity $O(l_3dN + l_3d^2)$.

In conlusion, **For training**, the overall time complexity is $O((l_1+l_2)dN(d + k) + l_3d^2N + (2d + 1)N)$ and the memory complexity is $O((l_1 + l_2)d(N + kd + k) + l_3dN + l_3d^2 + 4dN)$; **For inference**, the overall time complexity is $O((l_1 + l_2n)dN(d + k) + 2nN + l_3d^2N)$ and the memory complexity is $O((l_1 + l_2n)d(N + kd + k) + (2d + 1)nN + l_3dN + l_3d^2)$. After removing the constants, the time and memory complexity for training and inference is $O(N)$, which is **linear** to the number of nodes N. The linear complexities predict the potential scalability and efficiency of LIME. The scalability of LIME has been verified by Section 5.

D A BITTER LESSON FROM THE REAL WORLD: GRAPH CROSS-SUPERVISED LEARNING PROBLEM IS IMPORTANT

We would like to emphasize the importance and reality of the Graph Cross-Supervised Learning problem proposed by our paper here.

Precedents in general machine learning. The cross-supervised learning problem (also named mixed-supervised learning or weak-shot learning) is a well-defined problem in the general machine learning and computer vision domain. Specifically, previous works have explored similar settings in a wide range of tasks, such as object detection [10, 21, 58], fine-grained classification [2], semantic segmentation [58], and instance segmentation [11, 17]. Similarly, learning on graphs also faces a similar problem but is challenged by the uniqueness of graph data. Therefore, we first propose the graph cross-supervised learning problem and give a corresponding solution, i.e. LIME model.

Real-world graph data face graph cross-supervised learning problems. Real-world graph web applications like social networks and recommendation systems face the graph mixed-supervised learning problem with two realistic intuitive pieces of evidence. The first evidence is realistic graphs are growing rapidly (more new nodes with more new classes arriving): the real-world report [1] have shown social network BeReal's 1,200% increase in Gen Z usage As of April 2023; the report [2] shows that the number of Boomers who'd used TikTok grew by 164%. The second piece of evidence is realistic graphs are facing more noise like bots compared to before (label noise rate would increase): the real-world report [3] have shown that bots present on social graphs like Twitter at high intensities (20%); the report [4] cited Carnegie Mellon research indicating that 30 to 49 percent of accounts tweeting about the protests were bots. These two pieces of evidence indicate that, with real-world graphs increasing rapidly and more disinformation/noise spread, it's impossible to maintain accurate, high-quality, low-efficient annotations on new nodes with new classes compared to the beginning of annotation, especially for realistic graph applications, that maintained graph data with new node classes are labeled through cheap and easy-maintaining approaches like crowdsourcing, pseudo labels by clustering, and annotators without too much expertise, which leads to the label noise. Therefore, there exist imbalanced label noise rates between base class nodes (nearly no noise) and new class nodes (w. noise). We abstract this problem to the graph cross-supervised learning problem: base class nodes are labeled accurately while new class nodes are labeled with noises.

[1] Top Social Media Statistics And Trends Of 2023, *Forbes, May 18, 2023.*
[2] The Fastest Growing Social Media Platforms of 2023, *Hubspot, April 10, 2023.*
[3] Twitter Bots Poised to Spread Disinformation Before Election, *The New York Times, October 29, 2020.*
[4] Who's a Bot? Who's Not? *The New York Times, June 16, 2020.*