Active Learning for Graphs with Noisy Structures

Hongliang Chi
hc443@njit.edu
New Jersey Institute of Technology
USA

Cong Qi
cq5@njit.edu
New Jersey Institute of Technology
USA

Suhang Wang
szw494@psu.edu
Penn State University
USA

Yao Ma
yao.ma@njit.edu
New Jersey Institute of Technology
USA

Abstract

Graph Neural Networks (GNNs) have seen significant success in tasks like node classification, primarily dependent on the availability of abundant labeled nodes. However, the excessive cost of labeling large-scale graphs led to a focus on active learning methods on graphs, which aim for efficient data selection. While most methods assume reliable graph topology, real-world scenarios often present noisy graphs. Designing an active learning framework for noisy graphs is challenging, as selecting data for labeling and obtaining a clean graph are naturally interdependent: selecting high-quality data requires clean graph structure while cleaning noisy graph structure needs adequate labeled data. Considering the challenge mentioned above, we propose a robust active learning framework named `GALClean` that adopts an iterative approach to achieve data selection and graph purification simultaneously. Furthermore, we summarize `GALClean` as an instance of the Expectation-Maximization (EM) algorithm, which provides a theoretical understanding of the design and mechanisms in `GALClean` and naturally leads to an enhanced version `GALClean+`. Extensive experiments have demonstrated the effectiveness and robustness of our proposed method.

1 Introduction

Graph neural networks (GNNs) [21, 30, 33] have demonstrated significant potential in learning graph representation and thus facilitate the advancements of many graph-related applications including fraud detection [23, 37], recommender system [11, 17], drug discovery and repurposing [9, 14], and traffic prediction [8, 36]. Despite their unprecedented success, GNNs typically require a large number of labeled data, especially when dealing with large-scale graphs [35]. However, it is often time-consuming and labor-intensive to obtain high-quality labels. Consequently, recent efforts have been devoted to developing active learning (AL) algorithms tailored for graphs to efficiently acquire labels with low cost [12, 15, 22, 24, 39, 47]. Specifically, graph active learning algorithms aim to select a limited amount of the most valuable nodes for labeling, which is expected to reduce the labeling efforts while benefiting the training process the most. These graph active learning algorithms often extract and combine the key characteristics of nodes from both graph topology and node features [3, 22, 35, 47] which helps measure the value of nodes, leading to efficient and effective node selection strategies.

However, the majority of existing graph AL methods [3, 35, 42] are developed under the assumption that the underlying graph is noise-free, a condition that is rarely met in real-world applications [31]. Moreover, as suggested in [18], adversarial attacks on graphs could exacerbate the situation by introducing noisy edges that connect dissimilar nodes. The presence of this structural noise can significantly impair the performance of existing graph AL algorithms, as many of these algorithms rely on graph topology information for node selection. For example, AGE [3] directly utilizes node-centric topological properties, such as degree, to guide node selection, favoring nodes with higher degrees. Furthermore, AGE employs Graph Neural Networks (GNNs) to integrate topological information for learning node embeddings, which are subsequently used to identify the most representative nodes. The noisy graph structures may lead nodes with a larger degree to propagate inaccurate supervision signals more significantly. Also, the embeddings generated through GNNs might be infiltrated by noise, thereby failing to accurately reflect the representativeness of nodes. Our preliminary investigation (detailed in Section 3) shows that several advanced graph AL algorithms have seen a drop in performance in the presence of structural noise. Therefore, it is of great importance to develop AL algorithms that can handle the noises in graphs.

Given this damaging effect of the noisy structure on the graph AL methods, one natural idea to handle the noise in graphs is to adopt graph cleaning algorithms to obtain a cleaner graph first and then apply the existing graph AL algorithms on the cleaned graph. However, most existing graph cleaning methods such as Pro-GNN and RS-GNN [6, 18] require labels for the cleaning process, which are not available in the active learning setting. Although unsupervised graph cleaning algorithms such as GCN-Jaccard [34] and GCN-SVD [10] do exist, typically, they can only slightly mitigate the issue of the noisy graph structure.

In this paper, we focus on addressing a significant and practical issue that has been largely overlooked: the task of conducting efficient active learning on noisy graphs. We primarily face three challenges: (i) how to accurately select valuable nodes for labeling with the presence of noisy graph structures? (ii) how to purify the noisy graph structures with limited labeled data? (iii) how to manage the complex interdependence of the first two objectives, given that the success of each is mutually dependent on the successful completion of the other? To tackle these issues, we present a novel iterative **G**raph **A**ctive **L**earning and **Clean**ing (`GALClean`) framework that maximizes the synergy between node selection and graph cleaning. Specifically, to reduce the impact of structural noise on data selection, we utilize the most recently cleaned graph on training a well-designed representation-generating model to learn a latent representation space for data selection. Moreover, a robust node selection strategy, focusing on choosing nodes that are not only valuable for the downstream task but also resilient to structural noise, is applied for choosing nodes for labelling. In parallel, the

latent representations are also utilized to train an edge-predictor that aids in purifying the noisy graph. The newly cleaned graph reciprocates by assisting the node selection process in the next iteration with a cleaner topology. We further demonstrate that the iterative process in GALClean can be naturally interpreted as an instance of Stochastic Expectation Maximization (Stochastic EM) algorithm [1, 26, 46], which provides theoretical understanding and support for GALClean. Expanding upon this theoretical interpretation, we further introduce an enhanced framework GALClean+, which runs a few more iterations of EM algorithm after the labeling budget is exhausted. Extensive experiments have been done to show the effectiveness and robustness of our frameworks under different noise settings.

2 Problem Definition

A graph is denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ are the sets of nodes and edges, respectively. The connection can be also described as an adjacency matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$ with N denoting the number of nodes in $\mathcal{V}$. $\mathbf{A}_{ij}$ is the i, j-th element of $\mathbf{A}$ reflecting the strength of the connection between nodes v_i and v_j. Each node $v_i \in \mathcal{V}$ is associated with a d-dimensional feature vector $\mathbf{x}_i \in \mathbb{R}^d$. The features for all nodes can be summarized as $\mathbf{X} \in \mathbb{R}^{N\times d}$. Also, each node v_i has an underlying label $\mathbf{y}_i$. The graph $\mathcal{G}$ is assumed to contain some noises in graph structures. In particular, there is a certain proportion of edges in $\mathcal{E}$ which are heterophilous.

The objective is to select a limited number of nodes for labeling while also cleaning the graph such that a downstream GNN model trained with these labeled nodes and the cleaned graph, achieves strong performance—specifically, it should be capable of inferring unknown node labels with high accuracy. For this purpose, we are provided access to an oracle O that can supply the label of a given node within a limited budget of B. We are permitted to select a set of B nodes from a candidate pool $\mathcal{V}_{pool} \subset \mathcal{V}$ for labeling. We denote the set of selected nodes as $\mathcal{V}_l$ and the cleaned version of the graph as $\mathcal{G}'$. This process is initialized by creating a set of labeled nodes $\mathcal{V}_{initial}$, typically containing a few nodes from each class. The process to obtain these outputs can be described as follows.

$$\mathcal{V}_l, \mathcal{G}' = \mathcal{A}\left(\mathcal{G}, \mathbf{X}, \mathcal{V}_{pool}, O, \mathcal{V}_{initial}\right),$$

where $\mathcal{A}$ is the framework to be proposed in this paper.

3 Preliminary Analysis

The detrimental effects of structure noise for active learning on graphs and the consequent modeling step are two folded:

- **Data Effect**: Current graph active learning methods intensively rely on graph information to identify key nodes. The presence of noise edges can compromise the quality of the nodes selected.
- **Model Effect**: Graph Neural Networks (GNNs) utilize message-passing to aggregate information from neighboring nodes on graphs. Consequently, the training and inference of downstream GNNs could be significantly distorted if noise information propagates across nodes [48].

To examine how the noisy graph impacts the effectiveness of existing graph active learning methods, we conduct an empirical experiment on several recent advanced graph active learning models such as AGE [3], LSCALE [22], GRAIN [42] and ALG [39]. A brief introduction on these methods can be found in Section 5.1. Specifically, we first generate a perturbed graph by randomly adding edges between nodes from different classes. The number of noisy edges added equals the number of edges in the original graph. To clearly understand the *Data Effect* and *Model Effect* of the detrimental effects of structure noises. We execute each baseline under four Noise Scenarios related to the usage of the perturbed graph.

Table 1: Node classification performance under four different noise conditions.

Model	Noise Scenario	Cora	Citeseer	Pubmed
AGE	Noise-Free	77.07%	68.26%	76.52%
	Perturbed*	76.09%	67.21%	73.60%
	Perturbed**	52.74%	45.95%	54.91%
	Precleaned	57.50%	51.56%	56.95%
LSCALE	Noise-Free	78.54%	68.79%	78.39%
	Perturbed*	76.38%	65.87%	72.93%
	Perturbed**	50.31%	44.50%	53.48%
	Precleaned	57.12%	49.67%	55.22%
GRAIN	Noise-Free	78.42%	68.46%	78.27%
	Perturbed*	78.31%	66.36%	75.97%
	Perturbed**	53.76%	48.41%	56.61%
	Precleaned	61.26%	54.80%	56.83%
ALG	Noise-Free	77.68%	69.44%	78.66%
	Perturbed*	75.91%	68.15%	75.53%
	Perturbed**	52.10%	47.54%	56.81%
	Precleaned	58.73%	53.99%	59.45%

- Noise-Free: In this scenario, execute both the active learning and GCN model evaluation on the clean graph.
- Perturbed*: Here, we utilize the perturbed graph for graph active learning model. When it comes to training and testing the GCN model, we revert to using the original graph. This setup allows us to examine the *data effect* of the noisy structures, i.e, how the noisy structures compromise the quality of active learning.
- Perturbed**: In this case, we fully incorporate the perturbed graph into our experiment. Both the execution of the graph active learning models and the downstream GCN evaluation are conducted on this perturbed graph. As such, this scenario reveals the impacts of both the *data effect* and the *model effect*.
- Pre-cleaned: In this setup, we utilize a graph pre-processing technique known as Jaccard-GCN [34] to cleanse the noisy graph and generate a pre-cleaned graph. Both the graph active learning model and the subsequent GCN evaluation are conducted on this pre-cleaned graph.

Under all those scenarios, we run all active learning methods to select the same number of nodes for labeling. In particular, we fix the budget as $10 \times C$, with C denoting the number of classes. The comparative results of this empirical investigation are shown in Table 1. It is evident that the use of a perturbed graph in either the active learning step or the downstream GNN training and inference steps can strongly impair the models' performance. This highlights the importance and necessity for a robust graph active learning model that is capable of selecting high-quality nodes for labeling and producing a cleaner graph to facilitate the training and inference of the downstream GNNs. Also, under the Pre-cleaned scenario,

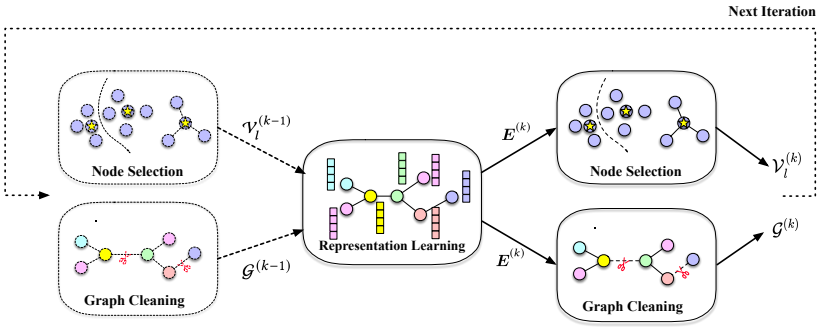

Figure 1: Overall framework of GALClean.

the active learning methods achieve better performance compared with the PERTURBED** setting. However, their performances are still significantly worse than those under the NOISE-FREE scenario, which demonstrates the incapability of the pre-cleaning technique without labeled data and motivates us to design a framework that performs graph cleaning and active learning simultaneously.

4 Methodology

As demonstrated in Table 1 of Section 3, the abnormal graph structures affect both the node selection and the training and inference of the downstream GNN, leading to unsatisfactory model performance. Supervised graph cleaning method is not even applicable from at the start of active learning due to label limitation, and unsupervised graph cleaning techniques offer only limited improvements. It's key to recognizing that graph cleaning and node selection are two interdependent tasks. While this interdependence introduces complexity when addressing either problem individually, it also presents significant potential for these tasks to iteratively enhance one another.

Motivated by the aforementioned observations, we propose an iterative framework GALClean, where node selection and graph cleaning are alternately executed upon the most recent information derived from the other task. At each iteration, batch-wise data selection is performed to select S nodes, concurrently with the graph cleaning step. Detailed procedures for this method are introduced in Section 4.1. This iterative process of GALClean enjoys a sound theoretical interpretation as an EM algorithm, treating the clean edges as the latent variable, as elaborated in Section 4.2. By subtly reformulating the EM algorithm, we extend GALClean to GALClean+ by leveraging selected labels and pseudo labels to continue the graph cleaning process. A description of GALClean+ is presented in Section 4.3.

4.1 GALClean Framework

At GALClean, we iterate over *representation learning*, *node selection*, and *graph cleaning* modules multiple times to gradually gather labeled nodes and purify the graph. A visual demonstration is provided at Figure 1. In essence, both the *node selection* and *graph cleaning* components aim to extract more information from the graph data. This is achieved by acquiring additional labeled data and mitigating noise within the graph structures from two perspectives. The information thus obtained is utilized to learn high-quality node representations in the *representation learning* component, which informs further iterations of *node selection* and *graph cleaning*.

To provide an overview of GALClean, we use the k-th iteration to briefly illustrate this process. Specifically, we define the labeled set and the graph after $(k-1)$-th iteration as $\mathcal{V}_l^{(k-1)}$ and $\mathcal{G}^{(k-1)}$, respectively. Then, in the n-th iteration, we first utilize $\mathcal{V}_l^{(k-1)}$ and $\mathcal{G}^{(k-1)}$ to learn node representations $\mathbf{E}^{(k)}$ by training a *representation learning* model. These representations $\mathbf{E}^{(k)}$ incorporate the additional gained information from $(k-1)$-th iteration. They are utilized to expand the labeled set to $\mathcal{V}_l^{(k)}$ in the *node selection* component and obtain a cleaner graph $\mathcal{G}^{(k)}$ in the *graph cleaning* component by utilizing reliable pseudo labels derived from them. $\mathcal{V}_l^{(k)}$ and $\mathcal{G}^{(k)}$ will then be utilized to conduct the $(k+1)$-th iteration of the process. To initialize this process, we set $\mathcal{V}_l^{(0)} = \mathcal{V}_{initial}$ and $\mathcal{G}^{(0)} = \mathcal{G}$. Note that $\mathcal{V}_{initial}$ and $\mathcal{G}$ are introduced in Section 2. After a total of K iterations, we exhaust the labeling budget and conclude with a labeled set of nodes $\mathcal{V}_l^{(K)}$ and a graph $\mathcal{G}^{(K)}$. The final set of labeled nodes, $\mathcal{V}_l^{(K)}$, is also the output of the framework, i.e., $\mathcal{V}_l = \mathcal{V}_l^{(K)}$.

The design focus of GALClean is to execute batch-wise active learning concurrent with graph cleaning, and pass the new acquired and clean information through iterations to maximize its benefits. In this section, we first describe the process of *representation learning* with structural noises. Then, we introduce the *node selection* and *graph cleaning* processes which take the learned representations as input.

4.1.1 Representation Learning The objective of the *representation learning* component is to incorporate the new information gained from the previous iteration to learn better quality representations, which supports the processes of *node selection* and *graph cleaning* in future iterations. Graph active learning methods often learn node representations using GCN models with supervision from labeled data [3, 12]. The GCN model incorporates both the supervision signals and graph structural information in a coupled manner. In particular, the structural information is captured through a forward feature aggregation process. For noisy graphs, a major limitation of this learning process is that it inevitably incorporates structural noises, leading to undesirable node representations. To address this issue, as inspired by [13, 16], we propose to decouple the process of

capturing the graph structural information and the label information. With the set of labeled nodes $\mathcal{V}_l^{(k-1)}$ and the graph $\mathcal{G}^{(k-1)}$ produced in the $(k-1)$-th iteration, the overall objective is as follows.

$$\mathcal{L} = \mathcal{L}_l(\mathcal{V}_l^{(k-1)}, \mathbf{E}^{(k)}) + \alpha \mathcal{L}_g(\mathcal{G}^{(l-1)}, \mathbf{E}^{(k)}), \tag{1}$$

$$\mathbf{E}^{(k)} = MLP_1(\mathbf{X}; \mathbf{W}_1^{(k)}) \tag{2}$$

where $\mathbf{E}^{(k)}$ denotes the representations produced by Multi-layer Perceptron (MLP) with the original node features $\mathbf{X}$ as input and $\mathbf{W}_1^{(k)}$ denotes all parameters of the MLP model in k-th iteration. The terms $\mathcal{L}_l$ and $\mathcal{L}_g$ capture label information and graph structural information respectively. The hyper-parameter α balances the two terms. Specifically, $\mathcal{L}_l$ is the classification loss of MLP.

$$\mathcal{L}_l = \sum_{v_i \in \mathcal{V}_l^{(k-1)}} \ell(\mathbf{y}_i, \mathbf{p}_i^{(k)}), \text{ with } \mathbf{p}_i^{(k)} = MLP_2(\mathbf{E}_i^{(k)}; \mathbf{W}_2^{(k)}) \tag{3}$$

where $\mathbf{p}_i^{(k)}$ denotes the vector of logits for node v_i obtained by transforming $\mathbf{E}_i^{(k)}$ through MLP_2, and $\ell(\cdot)$ is the cross entropy loss. To capture the graph structural information in $\mathcal{G}^{(k-1)}$, in general, we aim to pull nodes connected by clean edges to be close to each other while pushing non-connected nodes or those connected by noisy edges apart in the representation space. Hence, we adapt the neighborhood contrastive loss [16] to include the edge strength weights learned in the previous iteration. Specifically, the adjacency matrix $\mathbf{A}^{(k-1)}$ of the graph $\mathcal{G}^{(k-1)}$ contains these edge strength weights (the process to obtain $\mathbf{A}^{(k-1)}$ will be introduced in the *graph cleaning* process in Section 4.1.3. In particular, $A_{ij}^{(k-1)}$ is non-zero only when node v_i and v_j are connected and a larger value of $A_{ij}^{(k-1)}$ indicates a higher probability that the edge between them is "clean". The adapted neighborhood contrastive loss is as follows.

$$\mathcal{L}_g = -\sum_{v_i \in \mathcal{V}} \log \frac{\sum_{j=1}^{N} \mathbf{A}_{ij}^{(k-1)} \exp\left(\cos\left(\mathbf{E}_i^{(k)}, \mathbf{E}_j^{(k)}\right)/\tau\right)}{\sum_{v_m \in \mathcal{M}(v_i)} \exp\left(\cos\left(\mathbf{E}_i^{(k)}, \mathbf{E}_m^{(k)}\right)/\tau\right)} \tag{4}$$

where $\mathcal{M}(v_i)$ denotes a set negative samples, cos denotes the cosine similarity, and τ denotes the temperature parameter. With the numerator of Eq. (4), we push "clean" neighbors close in the representations space. On the other hand, we push v_i away from those negative samples in $\mathcal{M}(v_i)$. In practice, following [4, 20, 25], $\mathcal{M}(v_i)$ is randomly sampled from $\mathcal{V}$. When the dataset is relatively small, the entire set $\mathcal{V}$ can serve as the set of negative samples.

With the overall decoupled loss $\mathcal{L}$, the supervision signals are free from structural noises. This is because we could flexibly control the incorporation of graph structural information by adjusting α. In the extreme case when the graph is totally unreliable, the balancing parameter α can be set to 0, making the supervised signals learned in $\mathcal{L}_l$ free of the effect of structural noise. The parameters $\mathbf{W}^{(k)}$ are learned by minimizing the overall decoupled loss $\mathcal{L}$, which generates representations and predictions applied in processes of *node selection* and *graph cleaning*.

4.1.2 Node Selection In each iteration of node selection, we aim to select S nodes from $\mathcal{V}_{pool}$ that can represent $\mathcal{V}_{pool}$ in the best way. To achieve this goal, following FeatProp [35] and other core-set approaches [3, 12, 28, 39], we run the K-means clustering algorithm with S clusters utilizing the representations $\mathbf{E}^{(k)}$ produced by the *representation learning* process. Then, we aim to select one node from each cluster. A straightforward way to do this is to select the most representative node from each cluster (the one close to the centroid). However, if the selected representative nodes are surrounded by noisy edges, inaccurate supervision signals will be propagated over the graph, which may even mislead the mode training. As shown in the preliminary investigation in Section 3, the existence of structural noise could affect the performance of those representativeness-based active learning methods [3, 12, 35, 39]. Therefore, when selecting nodes, in addition to representativeness, we also care about the cleanliness of the neighborhood of candidate nodes. For example, as shown in Figure 2, node a has a cleaner neighborhood than b and thus a higher cleanliness score. Based on this, we propose a novel cleanliness score that measures the risk of a node being influenced by noisy graph structures. When selecting nodes for labeling, we consider both the node representativeness score and the cleanliness score. As for the example in Figure 2, assuming node a and b have similar representativeness scores, we prefer a over b.

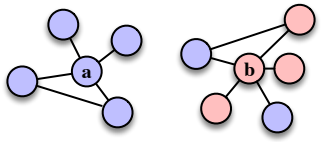

Figure 2: Node colors indicate classes; Node a has a cleaner neighborhood than node b.

As we have obtained a set of labeled nodes $\mathcal{V}^{(k-1)}$ from the previous iteration, we do not want to select these nodes and also some other nodes that are well represented by them. Thus, we first remove these nodes from the candidate pool. Next, we first describe the process of removing well-represented nodes. Then, we introduce the representativeness score and the robust cleanliness score. We conclude this section by describing how we use these metrics for selecting nodes for labeling.

Removing Well-Represented Nodes. Intuitively, nodes similar to ones in $\mathcal{V}_l^{(k-1)}$ should not be selected for labeling since they are already well-represented and will not provide rich additional information. Therefore, before running the formal node selection process, we need to remove nodes that are close to $\mathcal{V}_l^{(k-1)}$ from $\mathcal{V}_{pool}$. In particular, we model the distance between a node $v_i \in \mathcal{V}_{pool}$ and the set $\mathcal{V}_l^{(k-1)}$ as follows.

$$d(v_i, \mathcal{V}_l^{(k-1)}) = \min_{v_j \in \mathcal{V}_l^{(k-1)}} d(v_i, v_j), \tag{5}$$

where $d(v_i, v_j)$ measures the Euclidean distance between node v_i and v_j using representations $\mathbf{E}^{(k)}$. We rank all nodes in $\mathcal{V}_{pool}$ in a non-decreasing order according to their distance to $\mathcal{V}_l^{k-1}$ and remove the top $|\mathcal{V}_l^{(k-1)}| \cdot h$ of them. $h > 1$ is a hyper-parameter indicating how many nodes each labeled node covers. Note that nodes in $\mathcal{V}_l^{(k-1)}$ will always be removed as they have distance 0. After the removal, we denote the set of nodes left in the $\mathcal{V}_{pool}$ as the filtered candidate pool $\mathcal{V}_{filter}$.

Representativeness Score. Since we will select one node per cluster, for each node in $\mathcal{V}_{filter}$, we define a representativeness score for each cluster. We denote the centroid of the s-th cluster as c_s. For

node $v_i \in \mathcal{V}_{filter}$, its representativeness score corresponding to the s-th cluster is defined as follows.

$$r_{is} = 1/d(v_i, c_s), \tag{6}$$

where $d(v_i, c_s)$ measures the distance between v_i and c_s. Intuitively, nodes with smaller distances are considered more representative.

Cleanliness Score. Nodes that are connected with clean edges often share similar features. Hence, we define the cleanliness score for a node $v_i \in \mathcal{V}_{filter}$ based on the feature similarity to its neighbors as follows.

$$cl_i = \sum_{v_j \in \mathcal{N}(v_i)} cos(\mathbf{x}_n, \mathbf{x}_j), \tag{7}$$

where $\mathcal{N}(v_i)$ denotes the set of neighbors of node v_i and $cos(\mathbf{x}_n, \mathbf{x}_j)$ measures the cosine similarity between the original features of node v_i and v_j.

Node selection with Representativeness and Cleanliness Scores. In this part, we leverage the representativeness score and the cleanliness score together for node selection. Clearly, the representativeness score and the cleanliness score are at different scales and we care more about the relative relations between the candidate nodes in $\mathcal{V}_{filter}$. Hence, to combine these two scores, we rank the scores of candidate nodes and convert the scores into percentiles following [3, 43]. Specifically, for the i-th cluster, we rank all nodes in $\mathcal{V}_{filter}$ according to their representativeness score corresponding to the s-th cluster in a non-increasing order and obtain the percentile for each node. We denote the percentile for node $v_i \in \mathcal{V}_{filter}$ corresponding to i-th cluster as $\hat{r}_{is}$. Similarly, we obtain the percentile based on the cleanliness score, which is denoted as $\hat{cl}_i$ for node v_i. With these two percentiles, we select nodes for labeling as follows.

$$\mathcal{V}_{select}^{(k)} = \bigcup_{s=1}^{S} \{ \underset{v_i \in \mathcal{V}_{filter}}{\operatorname{argmin}} \ \hat{r}_{is} + \beta \cdot \hat{cl}_i \},$$

where β balances these two kinds of information, and we select the node with the largest combined score from each cluster. The labels of selected nodes $\mathcal{V}_{select}^{(k)}$ are then queried from the oracle $\mathcal{O}$. Finally, we include the newly selected node set $\mathcal{V}_{select}^{(k)}$ to the previous labeled set $\mathcal{V}_l^{(k-1)}$ expand the labeled set as follows.

$$\mathcal{V}_l^{(k)} = \mathcal{V}_{select}^{(k)} \bigcup \mathcal{V}_l^{(k-1)}$$

4.1.3 Graph Cleaning In this part, we aim to clean the graph structure by identifying and down-weighting the noisy edges in the graph. In particular, to achieve this goal, we propose to train an edge-predictor. However, building such an edge-predictor in an active learning setting is extremely challenging as labels are scarce. Specifically, it requires querying two nodes from the oracle to verify whether an edge is noisy or clean. In light of this challenge, we propose to construct a training set with pseudo labels of edges utilizing the representations $\mathbf{E}^{(k)}$. We then utilize this training set to train an edge-predictor, which is utilized for cleaning the graph. Next, we first describe the training set construction, then introduce details on utilizing the edge-predictor for graph learning.

Edge Training Set Construction. To produce pseudo labels for edges, we first produce probability logits for all nodes utilizing MLP_2 and $\mathbf{E}^{(k)}$ obtained from the *representation learning* component. We denote the vector of logits for node $v_i \in \mathcal{V}$ as $\mathbf{p}_i^{(k)}$. Note that for labeled nodes, we use their one-hot label vectors to replace the logits. We first obtain the pseudo labels for all nodes from the logits. Specifically, for each node v_i, the index corresponding to the largest dimension in the logits $\mathbf{p}_i^{(k)}$ is treated as the pseudo label, denoted as $\hat{y}_i$. Intuitively, we consider an edge is "clean" when its two nodes share the same label with high confidence. Therefore, the nodes in the following set are treated as positive samples for the edge predictor.

$$\mathcal{E}_+^{pseudo} = \{e_{ij} \in \mathcal{E} \mid v_i \in \mathcal{V}_{con}, v_j \in \mathcal{V}_{con}, \hat{y}_i = \hat{y}_j\}$$

where $\mathcal{V}_{con} = \{v_i \in \mathcal{V} \mid \mathbf{p}_i[\hat{y}_i] \geq \kappa\}$ denote the set of nodes with confident pseudo labels, and κ denotes a threshold of confident level. On the other hand, we consider an edge as "noisy" when its two nodes have different pseudo labels, which is formulated as follows.

$$\mathcal{E}_-^{pseudo} = \{e_{ij} \in \mathcal{E} \mid v_i \in \mathcal{V}, v_j \in \mathcal{V}, \hat{y}_i \neq \hat{y}_j\}.$$

Note that, we do not enforce the confidence constraint to the negative samples, since the edge is still highly likely to be negative even if the confidence of node pseudo labels is extremely high (larger than κ). In practical experiments, we tried to enforce the constraint, which turns out to affect the overall performance insignificantly.

Edge-predictor. With the training data defined in the previous part, we train an edge predictor. In particular, we model the probability of an edge being clean as follows.

$$p(e_{ij} = 1) = \sigma(\mathbf{z}_i^\top \mathbf{z}_j) \text{ with } \mathbf{z}_i = MLP_3(\mathbf{x}_i; \mathbf{W}_3^{(k)}), \tag{8}$$

where MLP_3 maps the original features into a representation space that is suitable for the probability estimation. The edge predictor is trained by maximizing the following probability.

$$P_{edge} = \prod_{e_{ij} \in \mathcal{E}_+^{pseudo}} p(e_{ij} = 1) \cdot \prod_{e_{ij} \in \mathcal{E}_-^{pseudo}} (1 - p(e_{ij} = 1)). \tag{9}$$

In practice, instead of maximizing P_{edge}, we minimize the negative of its logarithm. Once we obtain $\mathbf{W}_3^{(k)}$), we infer the probability $p(e_{ij} = 1)$ for all edges $e_{ij} \in \mathcal{E}$ and update the edge weights in the adjacency matrix as follows.

$$\mathbf{A}_{ij}^{(k)} = \begin{cases} p(e_{ij} = 1), & e_{ij} \in \mathcal{E} \\ 0, & \text{others.} \end{cases} \tag{10}$$

The probabilistic adjacency matrix $\mathbf{A}_{ij}^{(k)}$ defines a distribution for the discrete clean graph $\mathcal{G}$. In our case, we adopt the weighted graph defined by $\mathbf{A}_{ij}^{(k)}$ as $\mathcal{G}^{(k)}$ for the *representation learning* in the following iteration. Note that $\mathcal{G}^{(k)}$ is the expectation of the distribution defined by $\mathbf{A}_{ij}^{(k)}$.

4.2 GALClean as an EM algorithm

In this section, we show that the proposed GALClean framework can be understood from the perspective of an expectation-maximization (EM) algorithm. We first briefly introduce the concepts of the EM algorithm. Then, we explain how our framework can be formulated as an instance of the EM algorithm.

4.2.1 General Expectation Maximization algorithm An EM algorithm [7] is an iterative method used in machine learning for parameter estimation in probabilistic models that involve latent variables. The algorithm operates on a joint distribution $p(\mathbf{U}, \mathbf{z} \mid \boldsymbol{\theta})$ over the observed variable $\mathbf{U}$, the unobserved latent variables $\mathbf{z} \in \mathcal{Z}$ and model parameters $\boldsymbol{\theta}$. The goal is to maximize the likelihood function $p(\mathbf{U} \mid \boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$. Given N observations $\{\mathbf{u}_i\}_{i=1}^N$ of $\boldsymbol{X}$, The EM algorithm typically follows a two-step process:

E-step: The E-step is to compute the expected value of the likelihood function given the observed data, the posterior distribution of the latent variables, and updated parameters $\boldsymbol{\theta}^{\text{old}}$:

$$Q\left(\boldsymbol{\theta}; \boldsymbol{\theta}_{old}\right) = \frac{1}{N} \sum_{i=1}^{N} \int_{\mathcal{Z}} p\left(\mathbf{z} \mid \mathbf{u}_i\right) \cdot \log p\left(\mathbf{u}_i, \mathbf{z}\right) \mathrm{d}\mathbf{z}$$

M-step: The M-step is to update the parameters by maximizing the function $Q\left(\boldsymbol{\theta}; \boldsymbol{\theta}_{old}\right)$. Typically, the maximization in the M-step is conducted through gradient based-methods. When dealing with large datasets, obtaining the full gradient on all observations might be probability expensive. The Stochastic Expectation Maximization [1, 26, 46] was proposed to deal with such scenarios. Specifically, in the M-step of the stochastic EM, stochastic gradient-based methods are adopted for the maximization, where the gradient is estimated on a small batch of observations.

4.2.2 Interpret GALClean *as a Stochastic EM Algorithm* In our setting, the original graph structure is provided but with noisy edges. Since the clean graph structure is unknown, we treat the clean graph structure as the latent variable. Ideally, if we were able to observe labels for all nodes, they can serve as the observed variables in a standard EM algorithm. In this case, the goal of EM is to maximize the following marginal log-likelihood of all nodes in $\mathcal{V}$ with their corresponding labels observed:

$$\sum_{v_i \in \mathcal{V}} \ln p(\mathbf{y}_i, \mathbf{X} \mid \boldsymbol{\theta}) = \sum_{v_i \in \mathcal{V}} \int \ln p(\mathbf{y}_i, \mathbf{X}, \mathbb{G} \mid \boldsymbol{\theta}) d\mathbb{G}. \tag{11}$$

where $p(\mathbf{y}_i, \mathbf{X}, \mathbb{G}, \mid \boldsymbol{\theta})$ is the likelihood for node v_i with label $\mathbf{y}_i$ given the latent graph $\mathbb{G}$, and $\boldsymbol{\theta}$ corresponds to the model parameters. However, in our scenario, only a very small subset of nodes are observed with labels. Hence, we adopt stochastic gradient methods to optimize the log-likelihood in the M-step. In particular, the iterative process of GALClean can be regarded as an instance of a stochastic EM algorithm. We utilize the k-th iteration of the GALClean to illustrate the corresponding E and M-steps.

E-step: In the E-step, we aim to obtain the expectation of likelihood function given the observed data, the updated distribution of latent variables, and the updated parameters:

$$Q\left(\boldsymbol{\theta}; \boldsymbol{\theta}^{(k-1)}\right) = \sum_{v_i \in \mathcal{V}} \int p\left(\mathbb{G} \mid \mathbf{X}, \mathbf{y}_i, \boldsymbol{\theta}^{(k-1)}\right) \ln p(\mathbf{y}_i, \mathbf{X}, \mathbb{G} \mid \boldsymbol{\theta}) d\mathbb{G}, \tag{12}$$

where $\boldsymbol{\theta}^{(k-1)}$ refers to the model parameters estimated at the $(k-1)$-th iteration. $p\left(\mathbb{G} \mid \mathbf{X}, \mathbf{y}_i, \boldsymbol{\theta}^{(k-1)}\right)$ is the posterior distribution of the latent graph $\mathbb{G}$, which is described by the probabilistic adjacency matrix in Eq. (10). Computing the expectation in Eq. (12) is prohibitively expensive. Therefore, we approximate the entire posterior distribution using a Dirac delta function (δ distribution), a method known as variational approximation. The optimal δ distribution is defined at the Maximum a posteriori (MAP) of $\mathbb{G}$ [2]. In particular, in our case, the mass of δ distribution is concentrated at the expectation of the posterior distribution $\mathbb{G}^{(k-1)}$ (obtained in *graph cleaning*) and has 0 mass anywhere else. With the δ distribution, we approximate Eq. (12) as follows.

$$Q\left(\boldsymbol{\theta}; \boldsymbol{\theta}^{(k-1)}\right) = \sum_{v_i \in \mathcal{V}} \log p(\mathbf{y}_i, \mathbf{X}, \mathbb{G}^{(k-1)} \mid \boldsymbol{\theta}) \tag{13}$$

In conclusion, the E-step corresponds to the *graph cleaning* component in GALClean.

M-step: In the M-step, we aim to maximize Eq. (13). Due to the limited availability of labeled data. We optimized it and obtain the updated model parameters $\boldsymbol{\theta}^{(k)}$ with stochastic gradient estimated on $\mathcal{V}_l^{(k-1)}$. The M-step corresponds to the *representation learning* component in the GALClean framework. Specifically, $\boldsymbol{\theta}^{(k)}$ summarizes all parameters in Section 4.1.3 including $\mathbf{W}_1^{(k)}$ and $\mathbf{W}_2^{(k)}$. Note that the *data selection* also plays an important role in the M-step, as it gradually provides better $\mathcal{V}_l^{(k-1)}$ for a more accurate estimation of the full gradient.

Next, we explain how the *graph cleaning* and *data selection* (active learning) enhance each other from the perspective of the stochastic EM algorithm: (a) The EM algorithm guarantees that the log-likelihood value increases with each iteration until convergence, leading to a progressively improved fit of the representation model. This improvement ensures that both the active learning and graph cleaning processes are fully leveraged based on the most recent and reliable information obtained thus far, which leads to high-quality *data selection*; and (b) Moreover, since only a batch of observed data is used to optimize the likelihood function during the M-step, the gradient derived from the batch data may exhibit high variance, especially when the size of observations is small. The proposed *data selection* strategy focuses on selecting data with representativeness and cleanliness, which helps to obtain a more reliable mini-batch gradient that is better aligned with the one derived from fully labeled nodes with the clean graph. This alignment ensures that the Stochastic EM algorithm is updated in a more unbiased manner, thereby enhancing the overall effectiveness and accuracy of the model, which, in turn, helps the inference of the latent graph in the *graph cleaning* component (E-step).

4.3 GALClean+

As described in the previous section, GALClean can be understood as an instance of a stochastic EM algorithm. A natural idea to extend the GALClean is to run a few more iterations of EM even after the labeling budget is exhausted, which may help further clean the graph with the available information. Hence, we propose an enhanced version of GALClean named GALClean+. In particular, after K iterations of GALClean, we run out of the labeling budget and obtain $\mathcal{V}_l^{(K)}$. We then continue the EM algorithm for a few more iterations, where, in the M-step, we always use $\mathcal{V}_l^{(K)}$ to calculate the gradient. Another slight modification we made is that we also treat the unlabeled data $\mathcal{V}/\mathcal{V}_l^{(K)}$ as unobserved latent variables and involve them in the remaining EM iterations.

5 Experiments

In this section, we start by introducing the dataset, baselines, noise generation mechanism, and the experimental settings that have been

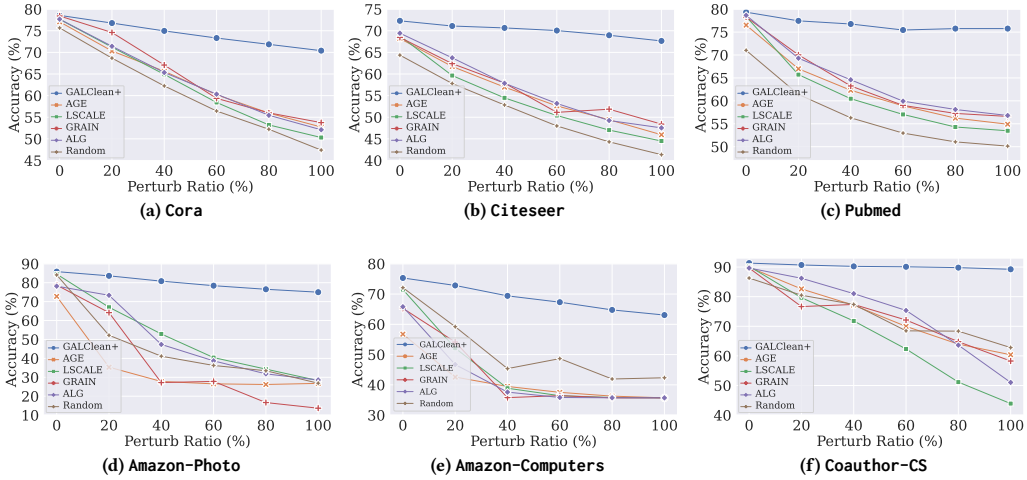

Figure 3: Active learning performance under random attacks

employed. Next, we present the performance of our framework, GALClean+, and various baselines under two distinct types of noise at varying levels. The primary objective of these experiments is to assess the effectiveness and robustness of our framework in diverse noisy scenarios. Lastly, we highlight the significance of key modules within our framework by conducting ablation studies, wherein we systematically remove key design components and report the impact on the overall performance.

5.1 Experimental settings

5.1.1 Dataset Our framework and baselines are evaluated on six datasets: Cora, Citeseer, Pubmed[27], Amazon-photo, Amazon-Computer, and Coauthor-CS [29]. The initial labeled set, $\mathcal{V}_{initial}$, is created by randomly sampling two nodes per class. We randomly select 50 nodes for the validation set, $\mathcal{V}_{valid}$, and 1000 nodes for the test set, $\mathcal{V}_{test}$. All methods select $8C$ nodes, $\mathcal{V}_{select}$, for labeling from the candidate pool, $\mathcal{V}_{pool}$, where C represents the number of classes, and $\mathcal{V}_{pool}$ encompasses nodes not included in $\mathcal{V}_{initial}$, $\mathcal{V}_{valid}$, or $\mathcal{V}_{test}$. Ultimately, the labeled set $\mathcal{V}_{labeled} = \mathcal{V}_{initial} \cup \mathcal{V}_{select}$ is employed to train a downstream GCN model [21]. We record the testing performance over $\mathcal{V}_{test}$. The experiments are performed 60 times using 10 different random initializations and six different random seeds. The report presents the average performance of the experiments.

5.1.2 Baselines We adopt the following classic and effective active learning models on graphs. (a) AGE [35] is a foundation active learning model on graphs, which combines several metrics including node centrality, node classification uncertainty, and node embedding representativeness to query nodes for labeling; (b) LSCALE [22] is a graph active learning model that utilizes a self-supervised learning method to construct an informative embedding space for node selection; (c)GRAIN [42] considers the magnitude of the node influence and the diversity of the influence simultaneously as the criterion to select nodes for labeling; (d)ALG [39] leverages a model-free representativeness measurement and a cost-effective informativeness measurement empowered by a decoupled GCN model to conduct data selection; (e) Random: We select nodes for labeling at random as the simplest strategy baseline.

5.1.3 Implementation Configuration Our framework is tuned on α, β and κ. The batch size S is 10 across all datasets. In assessing the GCN, we implement a two-layer GCN with a hidden dimension of 16 across all datasets, Amazon-Photo and Amazon-Computers being the exception. For Amazon-Photo and Amazon-Computers, we utilize a GCN model equipped with a hidden layer of 128 dimensions, as smaller hidden layers adversely impact the performance of the baselines.

5.1.4 Noisy Graph Generation To demonstrate the robustness of the proposed model against the structural noise, we randomly link two unconnected nodes from different classes to introduce a noisy edge into the graph. The number of newly added noisy edges increases from 0% of the number of edges (original graph) to 100% (highly perturbed graph) by a stride of adding 20% of the number of edges in the original graph. In addition, we leverage the recent state-of-the-art unsupervised attack model Contrastive Loss Gradient Attack (CLGA) [38] to generate attacked graph by adding 5%, 10%, 15%, and 20% additional noisy edges.

5.2 Main Results

In this section, we assess GALClean+'s performance under various types and levels of noise. We present and discuss the results under the *Random Attack Graphs* and *Unsupervised Adversarial Attacked Graphs* as follows.

5.2.1 Random Attacked Graphs The performances of our framework and baselines on graphs with random noises are shown in Figure 3. It can be observed from the table that our framework GALClean+ outperforms various baselines by a large margin on all 6 datasets, which indicates that the effectiveness of GALClean+ on selecting high-quality nodes for labeling while cleaning the graph.

5.2.2 Unsupervised Adversarial Attacked Graphs In this section, we investigate how robust GALClean+ is when graphs are perturbed by unsupervised adversarial attacks. The performances of GALClean+ and baselines are reported in Figure 4. As shown in the table, again,

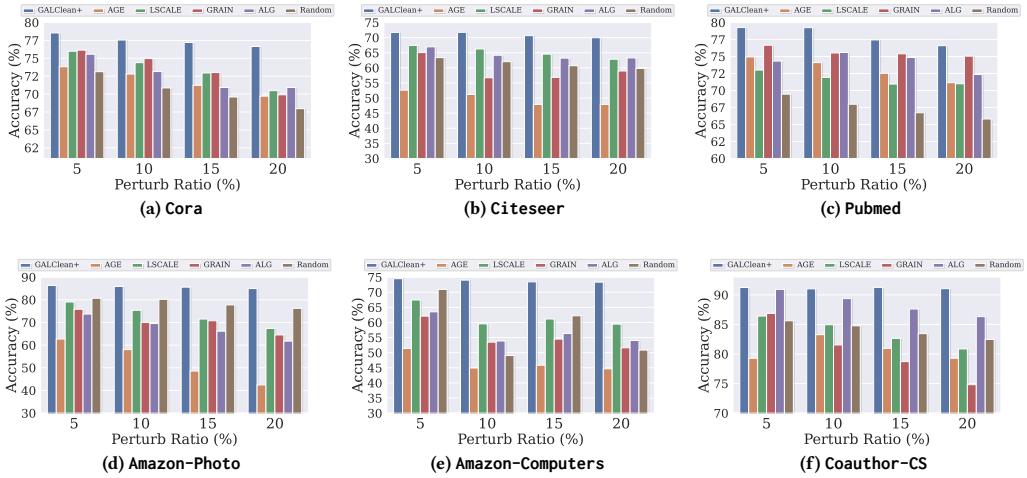

Figure 4: Active learning performance under Contrastive Loss Gradient Attack

GALClean+ has the best performance over all baselines on all 6 datasets. The performance demonstrates the superiority of GALClean+ even under a sophisticated unsupervised attack.

5.3 Ablation Study and Parameter Analysis

In this section, we scrutinize the design modules of GALClean+ through ablation study and parameter analysis. We report the results under 100% random noise setting on three citation datasets including Cora, Citeseer, and Pubmed as the observations on other settings and datasets are similar.

5.3.1 Effectiveness of the Proposed Node Selection Strategy To better understand how the proposed data selection Strategy, particularly the necessity of the node cleanliness metric introduced in Section 4.1.2. We run GALClean+ with and without the cleanliness score (C-score) applied in the data selection. All other parameters and settings were kept the same. The results are shown in Table 2. As we can observe from the table, on all three datasets, the cleanliness score helped improve the performance significantly, which shows the effectiveness of the proposed data selection strategy for noisy graphs.

Table 2: The effectiveness of cleanliness score (C-Score)

Setting	Cora	Citeseer	Pubmed
w/ C-Score	**70.40%±1.35%**	**67.65%±1.52%**	**75.76%±2.32%**
w/o C-Score	69.21%±1.47%	64.37%±2.68%	73.31%±3.91%

5.3.2 GALClean+ *vs* GALClean To verify the effectiveness of the additional EM iterations in GALClean+ as introduced in Section 4.3, we compare GALClean and GALClean+. The results are included in Table 3. As indicated in Table, GALClean+ consistently outperforms GALClean by a decent margin, which demonstrates the effectiveness of the additional EM iterations for producing a cleaner graph. However, the improvement is rather marginal, which also indicates that the additional EM iterations are not the most critical component of GALClean+.

Table 3: Performance of GALClean+ and GALClean

Model	Cora	Citeseer	Pubmed
GALClean+	**70.40%±1.35%**	**67.65%±1.52%**	**75.76%±2.32%**
GALClean	69.81%±1.14%	65.99%±1.62%	75.08%±2.29%

5.3.3 Effectiveness of the Pseudo Label Threshold in Section 4.1.3 In this part, we investigate how the parameter κ introduced for controlling the filtering threshold of pseudo labels affects the final performance. A higher κ means more pseudo labels will be filtered. In particular, we vary κ from 0 (without filtering out any pseudo labels) to 0.99. The results are demonstrated in Figure 5. The figure reveals that with $\kappa = 0$, the model's performance is subpar across all three datasets. This outcome underlines the essentiality of filtering less confident pseudo labels for training the edge predictor. As κ ascends, model performance generally improves due to the inclusion of more confident pseudo labels, thus providing more precise supervision. Upon further increasing κ, however, the model's performance begins to decline on the Cora and Citeseer datasets, attributed to the limited labeled data employed in training. Conversely, on PubMed, performance steadily escalates with κ up to 0.99. This pattern is predominantly owing to the relatively high confidence scores on PubMed, where over 52% of pseudo labels possess a confidence score exceeding 0.99. Therefore, a meticulous tuning of κ on PubMed may further enhance the proposed framework's performance.

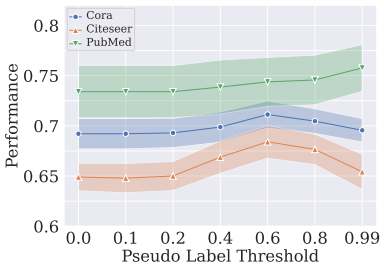

Figure 5: Parameter sensitivity analysis of κ of GALClean+

6 Related Work

Graph Active Learning. Active learning has been studied specifically for GNNs. AGE [3] mix multiple data selection metrics including information entropy, embedding representativeness, and graph centrality into its strategy. GPA [15] regards active learning as a sequential decision process on graphs and trains a GNN-based policy network to learn the optimal query strategy. ANRMAB [12] uses an active discriminative network representations with a multi-armed bandit mechanism for the graph active learning task. FeatProp [35] selects nodes for labeling in the representation space constructed by a parameter-free node feature propagation. LSCALE [22] exploits both labeled and unlabelled node representations for active learning on graphs. ALG [39] selects nodes for labeling by considering both node representativeness and informativeness and leverages decoupled GCNs to improve efficiency. GRAIN [42] performs data selection on graphs by achieving social influence maximization. RIM [40] converts node selection to a social influence maximization problem and also considers oracle noises. IGP [41] first proposes a soft-label approach to conduct active learning for GNNs. ALLIE [5] designs active learning specifically for large-scale imbalanced graph data. BIGENE [44] proposes a multi-agent Q-network consisting of a graph convolutional network module and a gated recurrent unit module for data selection. IGP [41] proposes the first GNN-based AL method suitable for soft labels. JuryGCN [19] quantifies the uncertainty of nodes with a frequentist-based approach and selects nodes based on uncertainties.

Graph Cleaning. Graph Structure Learning (GSL) aims to learn both a graph learning model and a graph structure simultaneously. GCN-Jaccard [34] remove edges according to the Jaccard similarity of node features. GCN-SVD [10] applies the low-rank approximation of the perturbed graph to reduce the effect of the adversarial attack. RGCN [45] reduces the impacts of adversarial attacks by introducing variance-based attention weight in the message-passing. Pro-GNN [18] learns the graph structure and the GNN model simultaneously considering the low rank and sparsity property of clean graphs. RS-GNN [6] mines information in the noisy graph as an additional supervision signal to obtain a cleaned graph, which helps to improve predictions of GNNs. GEN [32] optimizes a graph structure model and an observation model to gain the optimal graph in an iterative manner.

7 Conclusion

Current graph active learning methods universally rely on the utilization of accurate graph information to select high-quality nodes for labeling. However, structural noise is ubiquitous in real-world graphs. We first investigate how the edge noise deteriorates the performance of widely used graph active learning models. By identifying the challenges of performing active learning on noisy graphs, we propose a novel iterative graph active learning framework robust to graph noise by performing data selection and graph learning simultaneously. Not only high-quality data is selected at the end of the algorithm, but a cleaned graph will also be generated and utilized in the downstream graph tasks. Importantly, our framework has a solid theoretical interpretation as an EM algorithm. This interpretation provides a robust theoretical foundation. Extensive empirical experiment shows the proposed framework `GALClean+` has strong performance superiority over various baselines, especially when the noise is heavy.

References

[1] Sivaraman Balakrishnan, Martin J Wainwright, and Bin Yu. 2017. Statistical guarantees for the EM algorithm: From population to sample-based analysis. (2017).

[2] Matthew James Beal. 2003. *Variational algorithms for approximate Bayesian inference*. University of London, University College London (United Kingdom).

[3] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. 2017. Active learning for graph embedding. *arXiv preprint arXiv:1705.05085* (2017).

[4] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*. PMLR, 1597–1607.

[5] Limeng Cui, Xianfeng Tang, Sumeet Katariya, Nikhil Rao, Pallav Agrawal, Karthik Subbian, and Dongwon Lee. 2022. ALLIE: Active Learning on Large-scale Imbalanced Graphs. In *Proceedings of the ACM Web Conference 2022*. 690–698.

[6] Enyan Dai, Wei Jin, Hui Liu, and Suhang Wang. 2022. Towards robust graph neural networks for noisy graphs with sparse labels. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*. 181–191.

[7] Arthur P Dempster, Nan M Laird, and Donald B Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the royal statistical society: series B (methodological)* 39, 1 (1977), 1–22.

[8] Frederik Diehl, Thomas Brunner, Michael Truong Le, and Alois Knoll. 2019. Graph neural networks for modelling traffic participant interaction. In *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 695–701.

[9] Siddhant Doshi and Sundeep Prabhakar Chepuri. 2020. Dr-COVID: graph neural networks for SARS-CoV-2 drug repurposing. *arXiv preprint arXiv:2012.02151* (2020).

[10] Negin Entezari, Saba A Al-Sayouri, Amirali Darvishzadeh, and Evangelos E Papalexakis. 2020. All you need is low (rank) defending against adversarial attacks on graphs. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 169–177.

[11] Chen Gao, Xiang Wang, Xiangnan He, and Yong Li. 2022. Graph neural networks for recommender system. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*. 1623–1625.

[12] Li Gao, Hong Yang, Chuan Zhou, Jia Wu, Shirui Pan, and Yue Hu. 2018. Active discriminative network representation learning. In *IJCAI International Joint Conference on Artificial Intelligence*.

[13] Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997* (2018).

[14] Kanglin Hsieh, Yinyin Wang, Luyao Chen, Zhongming Zhao, Sean Savitz, Xiaoqian Jiang, Jing Tang, and Yejin Kim. 2021. Drug repurposing for COVID-19 using graph neural network and harmonizing multiple evidence. *Scientific reports* 11, 1 (2021), 1–13.

[15] Shengding Hu, Zheng Xiong, Meng Qu, Xingdi Yuan, Marc-Alexandre Côté, Zhiyuan Liu, and Jian Tang. 2020. Graph policy network for transferable active learning on graphs. *Advances in Neural Information Processing Systems* 33 (2020), 10174–10185.

[16] Yang Hu, Haoxuan You, Zhecan Wang, Zhicheng Wang, Erjin Zhou, and Yue Gao. 2021. Graph-MLP: node classification without message passing in graph. *arXiv preprint arXiv:2106.04051* (2021).

[17] Tinglin Huang, Yuxiao Dong, Ming Ding, Zhen Yang, Wenzheng Feng, Xinyu Wang, and Jie Tang. 2021. Mixgcf: An improved training method for graph neural network-based recommender systems. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 665–674.

[18] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. 2020. Graph structure learning for robust graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 66–74.

[19] Jian Kang, Qinghai Zhou, and Hanghang Tong. 2022. JuryGCN: quantifying jackknife uncertainty on graph convolutional networks. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 742–752.

[20] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. 2020. Supervised contrastive learning. *Advances in neural information processing systems* 33 (2020), 18661–18673.

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

[22] Juncheng Liu, Yiwei Wang, Bryan Hooi, Renchi Yang, and X. Xiao. 2020. LSCALE: Latent Space Clustering-Based Active Learning for Node Classification.

[23] Yang Liu, Xiang Ao, Zidi Qin, Jianfeng Chi, Jinghua Feng, Hao Yang, and Qing He. 2021. Pick and choose: a GNN-based imbalanced learning approach for fraud detection. In *Proceedings of the Web Conference 2021*. 3168–3177.

[24] Kaushalya Madhawa and Tsuyoshi Murata. 2020. Active learning for node classification: an evaluation. *Entropy* 22, 10 (2020), 1164.

[25] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748* (2018).

[26] George Papandreou, Liang-Chieh Chen, Kevin P Murphy, and Alan L Yuille. 2015. Weakly-and semi-supervised learning of a deep convolutional network for semantic image segmentation. In *Proceedings of the IEEE international conference on computer vision*. 1742–1750.

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

[28] Ozan Sener and Silvio Savarese. 2017. Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489* (2017).

[29] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of Graph Neural Network Evaluation. *Relational Representation Learning Workshop, NeurIPS 2018* (2018).

[30] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[31] Dan J Wang, Xiaolin Shi, Daniel A McFarland, and Jure Leskovec. 2012. Measurement error in network data: A re-classification. *Social Networks* 34, 4 (2012), 396–409.

[32] Ruijia Wang, Shuai Mou, Xiao Wang, Wanpeng Xiao, Qi Ju, Chuan Shi, and Xing Xie. 2021. Graph structure estimation neural networks. In *Proceedings of the Web Conference 2021*. 342–353.

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

[34] Huijun Wu, Chen Wang, Yuriy Tyshetskiy, Andrew Docherty, Kai Lu, and Liming Zhu. 2019. Adversarial examples on graph data: Deep insights into attack and defense. *arXiv preprint arXiv:1903.01610* (2019).

[35] Yuexin Wu, Yichong Xu, Aarti Singh, Yiming Yang, and Artur Dubrawski. 2019. Active learning for graph neural networks via node feature propagation. *arXiv preprint arXiv:1910.07567* (2019).

[36] Yi Xie, Yun Xiong, and Yangyong Zhu. 2020. SAST-GNN: a self-attention based spatio-temporal graph neural network for traffic prediction. In *International Conference on Database Systems for Advanced Applications*. Springer, 707–714.

[37] Yufan Zeng and Jiashan Tang. 2021. Rlc-gnn: An improved deep architecture for spatial-based graph neural network with application to fraud detection. *Applied Sciences* 11, 12 (2021), 5656.

[38] Sixiao Zhang, Hongxu Chen, Xiangguo Sun, Yicong Li, and Guandong Xu. 2022. Unsupervised graph poisoning attack via contrastive loss back-propagation. In *Proceedings of the ACM Web Conference 2022*. 1322–1330.

[39] Wentao Zhang, Yu Shen, Yang Li, Lei Chen, Zhi Yang, and Bin Cui. 2021. ALG: fast and accurate active learning framework for graph convolutional networks. In *Proceedings of the 2021 International Conference on Management of Data*. 2366–2374.

[40] Wentao Zhang, Yexin Wang, Zhenbang You, Meng Cao, Ping Huang, Jiulong Shan, Zhi Yang, and Bin Cui. 2021. Rim: Reliable influence-based active learning on graphs. *Advances in Neural Information Processing Systems* 34 (2021), 27978–27990.

[41] Wentao Zhang, Yexin Wang, Zhenbang You, Meng Cao, Ping Huang, Jiulong Shan, Zhi Yang, and Bin Cui. 2022. Information Gain Propagation: a new way to Graph Active Learning with Soft Labels. *arXiv preprint arXiv:2203.01093* (2022).

[42] Wentao Zhang, Zhi Yang, Yexin Wang, Yu Shen, Yang Li, Liang Wang, and Bin Cui. 2021. Grain: Improving data efficiency of graph neural networks via diversified influence maximization. *arXiv preprint arXiv:2108.00219* (2021).

[43] Ye Zhang, Matthew Lease, and Byron Wallace. 2017. Active discriminative text representation learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 31.

[44] Yuheng Zhang, Hanghang Tong, Yinglong Xia, Yan Zhu, Yuejie Chi, and Lei Ying. 2022. Batch active learning with graph neural networks via multi-agent deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 9118–9126.

[45] Dingyuan Zhu, Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2019. Robust graph convolutional networks against adversarial attacks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 1399–1407.

[46] Rongda Zhu, Lingxiao Wang, Chengxiang Zhai, and Quanquan Gu. 2017. High-dimensional variance-reduced stochastic gradient expectation-maximization algorithm. In *International Conference on Machine Learning*. PMLR, 4180–4188.

[47] Yanqiao Zhu, Weizhi Xu, Qiang Liu, and Shu Wu. 2020. When contrastive learning meets active learning: A novel graph active learning paradigm with self-supervision. *arXiv preprint arXiv:2010.16091* (2020).

[48] Yanqiao Zhu, Weizhi Xu, Jinghao Zhang, Yuanqi Du, Jieyu Zhang, Qiang Liu, Carl Yang, and Shu Wu. 2021. A Survey on Graph Structure Learning: Progress and Opportunities. *arXiv e-prints* (2021), arXiv–2103.