Active Learning for Graphs with Noisy Structures

Hongliang Chi* Cong Qi† Suhang Wang‡ Yao Ma§

Abstract

Graph Neural Networks (GNNs) have seen significant success in tasks such as node classification, largely contingent upon the availability of sufficient labeled nodes. Yet, the excessive cost of labeling large-scale graphs led to a focus on active learning on graphs, which aims for effective data selection to maximize downstream model performance. Notably, most existing methods assume reliable graph topology, while real-world scenarios often present noisy graphs. Given this, designing a successful active learning framework for noisy graphs is highly needed but challenging, as selecting data for labeling and obtaining a clean graph are two tasks naturally interdependent: selecting high-quality data requires clean graph structure while cleaning noisy graph structure requires sufficient labeled data. Considering the complexity mentioned above, we propose an active learning framework, `GALClean`, which has been specifically designed to adopt an iterative approach for conducting both data selection and graph purification simultaneously with best information learned from the prior iteration. Importantly, we summarize `GALClean` as an instance of the Expectation-Maximization algorithm, which provides a theoretical understanding of its design and mechanisms. This theory naturally leads to an enhanced version, `GALClean+`. Extensive experiments have demonstrated the effectiveness and robustness of our proposed method across various types and levels of noisy graphs.

Keywords: Graph Neural Networks, Active Learning, Noisy Learning.

1 Introduction

Graph Neural Networks [17,26] have demonstrated great potential in learning graph representation and thus facilitate the advancements of many graph-related applications including fraud detection [19,30], recommender system [9,14], and traffic prediction [8, 29]. Despite their success, GNNs typically require a large number of labeled data, especially when dealing with large-scale graphs [28]. However, it is often expensive to obtain high-quality labels. Recent efforts have been devoted to developing active learning (AL) algorithms for graphs to efficiently acquire labels with low cost [10, 12, 18, 20, 31]. Specifically, graph active learning (GAL) aims to select a limited number of nodes for labeling, which is expected to reduce the labeling efforts by enhancing the downstream GNNs performance as much as possible. These GAL algorithms often extract and leverage the key information of nodes from both graph topology and features [3, 18, 28], to measure the importance of nodes and thereby perform effective node selection.

However, the most existing GAL methods [3,28,34] are developed under the assumption that the underlying graph is noise-free, a condition that is rarely met in real-world applications [24]. Moreover, as suggested in [15], adversarial attacks on graphs could exacerbate the situation by introducing noisy edges that connect dissimilar nodes. This structural noise can compromise GAL performance, as our *Preliminary Analysis* in Section 3 reveals. One potential solution involves cleaning the graph prior to applying GAL algorithms. However, conventional graph cleaning methods such as Pro-GNN and RS-GNN [6, 15] require labels for the cleaning process, which are not available in the active learning setting. Although unsupervised graph cleaning algorithms such as GCN-Jaccard [27] 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, `GALClean` leverages a purified graph to train a representation model to learn node

*Rensselaer Polytechnic Institute, chih3@rpi.edu
†New Jersey Institute of Technology, cq5@njit.edu
‡Penn State University, szw494@psu.edu
§Rensselaer Polytechnic Institute, may13@rpi.edu

representations 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 on the trustworthy node representations yielded prior. In parallel, those reliable representations are used to train an edge-predictor for producing a cleaner graph. This graph again reciprocates by assisting the node selection process in the next iteration with less noises. We discover that the iterative process in `GALClean` can be naturally interpreted as an instance of Stochastic Expectation Maximization (Stochastic EM) algorithm [1, 22, 38], 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 of `GALClean+` and also the importance of each key design component.

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. The connection is 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 cleaning the graph such that a downstream GNN model trained with these labeled nodes and the cleaned graph, achieves strong performance. For this purpose, we are provided access to an oracle $\mathcal{O}$ 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 $\mathcal{V}_l, \mathcal{G}' = \mathcal{A}(\mathcal{G}, \mathbf{X}, \mathcal{V}_{pool}, \mathcal{O}, \mathcal{V}_{initial})$, where $\mathcal{A}$ is a graph active learning (GAL) model.

3 Preliminary Analysis

The detrimental effects of structure noises on GAL and the consequent modeling step are two folded:

- **Data Effect**: Current GAL methods intensively rely on graph information to identify key nodes. The presence of noise edges can compromise the quality of the nodes selected by those methods.
- **Model Effect**: 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 [39].

To examine how the noisy graph impacts the effectiveness of existing GAL methods, we conduct empirical experiments on several recent advanced graph active learning models such as `AGE` [3], `LSCALE` [18], `GRAIN` [34] and `ALG` [31]. A brief introduction to these methods can be found in Section 1 of the **supplementary file**. 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 structure noises. We run each baseline under four Noise Scenarios.

- NOISE-FREE: Both the active learning and GCN model evaluation are performed on the clean graph.
- PERTURBED*: A perturbed graph is used for active learning but the GCN model is trained and tested on a clean graph. This setup allows us to examine the *data effect* of the noisy structures.
- PERTURBED**: The active learning and the GCN modeling are both performed on the perturbed graph. As such, this scenario reveals the impacts of both the *data effect* and the *model effect*.
- PRE-CLEANED: A graph pre-processing method, Jaccard-GCN [27], is applied to cleanse the noisy graph and generate a pre-cleaned graph. Both the graph active learning model and the GCN modeling are conducted on this pre-cleaned graph.

Under those cases, GAL baselines is set to choose the same number of nodes for labeling. The detailed results of this empirical investigation are shown in Table 1. As shown in the results, it is evident that the use of a perturbed graph in either the active learning step or the GNN training and inference steps can strongly impair the models' performance. This highlights the necessity for a robust GAL model capable of selecting high-quality nodes in a noisy setting and also producing a cleaner graph to facilitate the modelling of downstream GNNs. Notably, under the PRE-CLEANED scenario, GAL methods achieve better performance compared with the PERTURBED** setting. However, their performances are still significantly worse than those under the NOISE-FREE scenario.

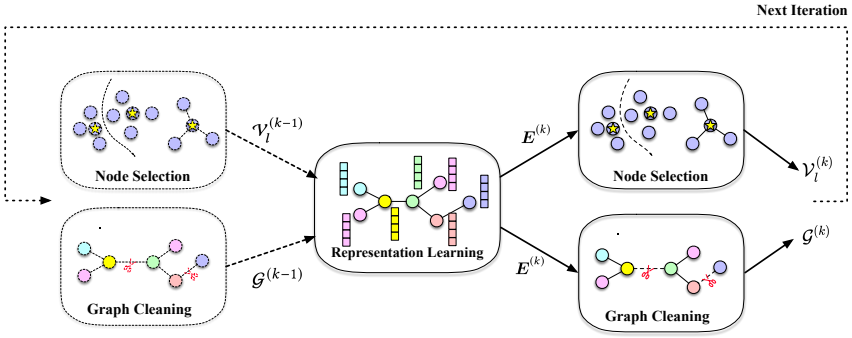


Figure 1: Overall framework of GALClean.

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%

4 Methodology

Given the observed **Data Effect** and **Model Effect**, a framework that can concurrently address graph cleaning and node selection is highly needed. Following this lead, we propose GALClean, a solution that not only achieves both node selection and graph cleaning but also ensures they mutually help each other's effectiveness.

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 k-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., $\mathcal{V}_l = \mathcal{V}_l^{(K)}$.

4.1.1 Representation Learning

The *representation learning* module in GALClean is designed to learn trustworthy node representations as the input of *node selection* and *graph cleaning* modules. Current GAL methods often learn node representations using GNNs with supervision from labeled data [3,10]. However, the GCN incorporates both the supervision signals and graph structural information in a coupled manner. For noisy graphs, a major limitation of this coupled design is that it inevitably includes structural noises, leading to undesirable node representations. To address this issue, as inspired by [11,13], 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.

$$\mathcal{L} = \mathcal{L}_l(\mathcal{V}_l^{(k-1)}, \mathbf{E}^{(k)}) + \alpha \mathcal{L}_g(\mathcal{G}^{(l-1)}, \mathbf{E}^{(k)}),$$
$$\mathbf{E}^{(k)} = MLP_1(\mathbf{X}; \mathbf{W}_1^{(k)}),$$

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 clas-

sification loss of MLP.

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

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 $\mathcal{G}^{(k-1)}$, we adapt the neighborhood contrastive loss [13] 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 nonzero 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.

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

where $\mathcal{M}(v_i)$ denotes a set negative samples, cos denotes the cosine similarity, and τ denotes the temperature parameter. In practice, following [4, 16, 21], $\mathcal{M}(v_i)$ 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 combined loss $\mathcal{L}$, the supervision signals are more robust to structural noises since graph information can be adaptively adjusted with 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

Here, `GALClean` is expected to select S nodes from $\mathcal{V}_{pool}$ that can represent $\mathcal{V}_{pool}$ in the best way. To achieve this, following FeatProp [28] and other coreset approaches [3,10,23,31], we run the K-means 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. Hence, in addition to representativeness, we also care about the cleanliness of the neighborhood of candidate nodes. 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 we have obtained a set of labeled nodes $\mathcal{V}^{(k-1)}$ from the previous iteration, we aim to avoid reselecting them or any nodes that are adequately represented by this set. 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 node selection.

Removing Well-Represented Nodes. Clearly, nodes similar to ones in $\mathcal{V}_l^{(k-1)}$ should not be selected since they are already well-represented and will not provide further 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.

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

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 `GALClean` selects one node per cluster, for each node in $\mathcal{V}_{filter}$, we define a representativeness score for each cluster. Let the centroid of the s-th cluster be denoted as c_s. For node $v_i \in \mathcal{V}_{filter}$, its representativeness score corresponding to the s-th cluster is defined as $r_{is} = 1/d(v_i, c_s)$, 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 $cl_i = \sum_{v_j \in \mathcal{N}(v_i)} cos(\mathbf{x}_n, \mathbf{x}_j)$, 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 their original features.

Node selection with Representativeness and Cleanliness Scores. Next, `GALClean` leverages the representativeness score and the cleanliness score together for node selection. Clearly, the representative-

ness 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, 35]. 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 $\mathcal{V}_l^{(k)} = \mathcal{V}_{select}^{(k)} \bigcup \mathcal{V}_l^{(k-1)}$.

4.1.3 Graph Cleaning

In this part, `GALClean` is designed to clean the graph structure by identifying and down-weighting the noisy edges in the graph with an edge-predictor. However, building such an edge-predictor in the AL 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, and 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 to be "clean" when its two nodes share the same label with high confidence. Therefore, the nodes in the following set are treated as positive samples.

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

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 confidence. On the other hand, we consider an edge as "noisy" when its two nodes have different pseudo labels, as formulated below.

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

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). We tried to enforce the constraint, which turned out to affect the overall performance insignificantly.

Edge-predictor. With the training data defined previously, we train an edge-predictor. The probability of an edge being clean is modeled as follows.

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

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.

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

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.

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

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 $\mathbf{A}_{ij}^{(k)}$.

5 `GALClean` as an EM algorithm

In this section, we show that the `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.

5.1 EM algorithm

An EM [7] is an iterative method used in machine learning for parameter estimation in probabilistic models that involve latent variables. The EM 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 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 $\boldsymbol{\theta}^{\text{old}}$

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

M-step: The M-step is to update the parameters by maximizing the function $\mathcal{Q}\left(\boldsymbol{\theta}; \boldsymbol{\theta}_{old}\right)$. Typically, the maximization in the M-step is conducted through gradient-based methods. In large-scale settings, computing the full gradient can be computationally costly. Stochastic Expectation Maximization [1, 22, 38] addresses this challenge by employing stochastic gradient methods, where the gradient is estimated using a subset of the data.

5.1.1 Interpret GALClean as a Stochastic EM

In our setting, the original graph structure is provided but with noisy edges. Since the clean graph structure is unknown, we treat it as the latent variable. Ideally, if we were able to observe labels for all nodes, they could serve as the observed variables in a standard EM. 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:

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

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 case, 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. 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:

$$\mathcal{Q}\left(\boldsymbol{\theta}; \boldsymbol{\theta}^{(k-1)}\right) = \sum_{v_i \in \mathcal{V}} \int p\left(\mathbb{G} \mid \mathbf{X}, \mathbf{y}_i, \boldsymbol{\theta}^{(k-1)}\right) \ln p(\mathbf{y}_i, \mathbf{X}, \mathbb{G} \mid \boldsymbol{\theta}) d\mathbb{G}, \quad (2)$$

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. (1). Computing the expectation in Eq. (2) 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. (2) as follows.

$$\mathcal{Q}\left(\boldsymbol{\theta}; \boldsymbol{\theta}^{(k-1)}\right) = \sum_{v_i \in \mathcal{V}} \log p(\mathbf{y}_i, \mathbf{X}, \mathbb{G}^{(k-1)} \mid \boldsymbol{\theta}) \quad (3)$$

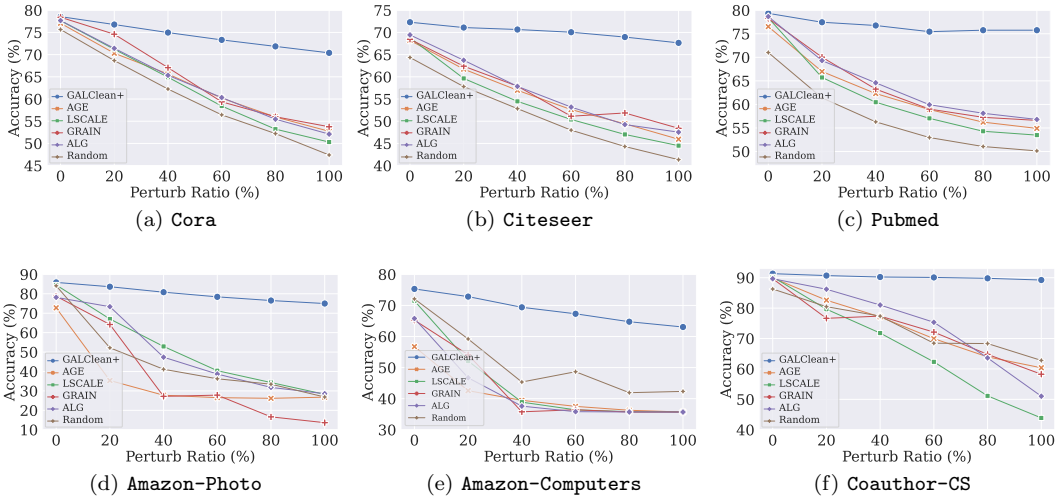
In conclusion, the E-step corresponds to the *graph cleaning* component in GALClean.

M-step: In the M-step, we maximize Eq. (3). Due to the limited labeled data, we optimize it and obtain the updated 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* enhance each other from the perspective of the stochastic EM algorithm: (a) The EM 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 AL 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 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).

5.1.2 GALClean+

As described in the previous section, GALClean can be understood as an instance of a stochastic EM. 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. Hence, we propose an enhanced version of GALClean



(a) Cora (b) Citeseer (c) Pubmed

(d) Amazon-Photo (e) Amazon-Computers (f) Coauthor-CS

Figure 2: Active Learning Performance under Random Edge-Adding Attacks

named GALClean+. In particular, after K iterations of GALClean, we run out of the labeling budget and obtain $\mathcal{V}_l^{(K)}$. We continue the EM for a few more iterations, where, in the M-step, we always use $\mathcal{V}_l^{(K)}$ to calculate the gradient. We also treat the unlabeled data $\mathcal{V}/\mathcal{V}_l^{(K)}$ as unobserved latent variables and involve them in the remaining EM iterations.

In the end, we present a detailed analysis of time complexity for GALClean and GALClean+ in Section 2 of the **supplementary file**.

6 Experiments

In this section, we conduct experiments to assess the effectiveness and robustness of our framework in two noisy scenarios or attacking cases. After that, we test and highlight the significance of its key design modules by conducting ablation studies and parameter analyses. Notably, we include the details of the datasets, baselines, graph noise/attack generation mechanisms, and experimental settings in the **supplementary materials**.

6.1 Main Results

Here, we assess GALClean+'s performance under various types and levels of attacks. In particular, we adopt *Random Edge-Adding Attack* and *Unsupervised Adversarial Attack* to perturb graphs at different levels. The perturbation process of these attacks can be found in Section 1 of supplementary file. Next, We present the major results of GALClean+ against comprehensive baselines with a thorough analysis.

Random Edge-Adding Attacked Graphs. The results on graphs with $n\%$ random edge-adding noises are shown in Figure 2. It can be seen that GALClean+ outperforms various baselines by a large margin on all six datasets, which indicates the effectiveness of GALClean+ on selecting high-quality nodes while cleaning the graph.

Unsupervised Adversarial Attacked Graphs. In this part, we investigate how robust GALClean+ is when graphs are perturbed by $n\%$ unsupervised adversarial attacks. The testing accuracy of GALClean+ and baselines are reported in Figure 3. Once again, GALClean+ outperformed all baselines, which demonstrates the superiority of GALClean+ even under a sophisticated unsupervised attack.

6.2 Ablation Study and Parameter Analysis

In this subsection, we scrutinize the design modules of GALClean+ through ablation study and parameter analysis. We report the results under 100% random edge-adding noise setting on citation datasets as the observations on other settings and datasets are similar. Additionally, we investigated how GALClean+ performs with limited labelling budgets. This budget sensitivity analysis is attached in Section 3 of the **supplementary file**.

6.2.1 Effectiveness of the Cleanliness-based Node Selection Strategy

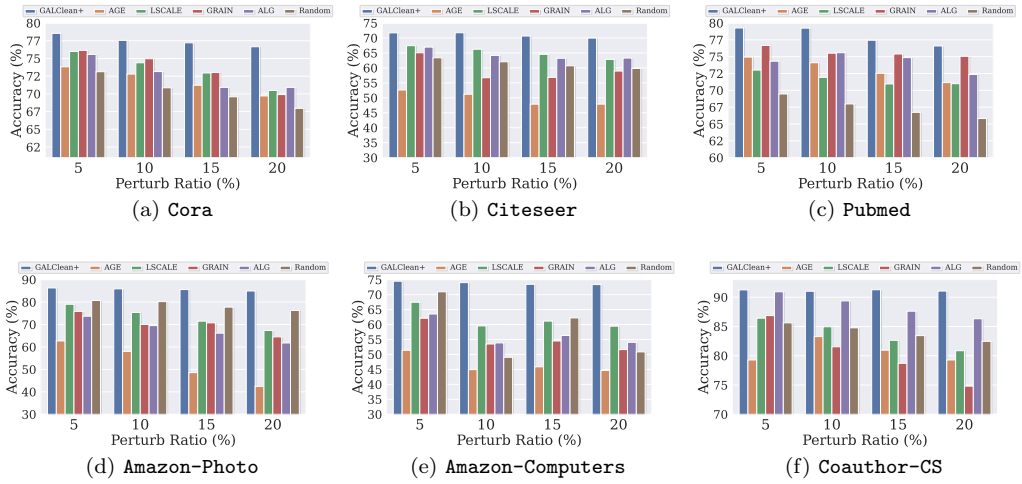
For better understanding how the proposed selection strategy, particularly the necessity of the node cleanliness metric introduced in Section 4.1.2, we run GALClean+ with and without the cleanliness score (C-score). All other parameters and settings were kept the same. The results are shown in Table 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 cleanliness-based data selection strategy for active learning tasks on noisy graphs.

Table 2: Impact of the Use of Cleanliness Score on Testing Accuracy (%) (↑↑: increase > 1%)

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

6.2.2 GALClean+ versus. GALClean

To clearly illustrate the improvement from the additional EM iterations in GALClean+ (introduced in Section 5.1.2), we compare the performance of GALClean and GALClean+ side by side, which is summarized in Table 3. The results show that GALClean+



(a) Cora (b) Citeseer (c) Pubmed

(d) Amazon-Photo (e) Amazon-Computers (f) Coauthor-CS

Figure 3: Active Learning Performance under Unsupervised Adversarial Attacks

consistently outperforms GALClean, which confirms the use of extra EM iterations in purifying graph structures and enhancing data selection quality.

Table 3: Tesing Accuracy (%) of GALClean+ and GALClean (↑: increase > 0.5%; ↑↑: increase > 1%)

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

6.2.3 Effectiveness of Thresholding with κ

In this part, we investigate how the parameter κ introduced in Section 4.1.3 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 4. 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 improve the performance.

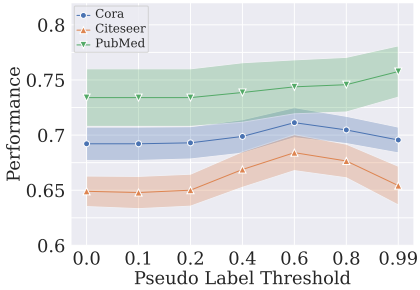


Figure 4: Parameter sensitivity analysis for κ

7 Related Work

Graph Active Learning (GAL). AL has been studied specifically for GNNs. AGE [3] mix multiple data selection metrics into its strategy. GPA [12] regards AL as a sequential decision process on graphs and trains a GNN-based policy network to learn the optimal query strategy. ANRMAB [10] uses an active discriminative network representations with a multi-armed bandit mechanism for the GAL task. FeatProp [28] selects nodes for labeling in the representations constructed by a parameter-free node feature propagation. LSCALE [18] exploits both labeled and unlabelled node representations for AL on graphs. ALG [31] selects nodes by considering both node representativeness and informativeness and leverages decoupled GCNs to improve efficiency. GRAIN [34] performs data selection on graphs by achieving social influence maximization. RIM [32] converts node selection to a social influence maximization problem and considers oracle noises. IGP [33] first proposes a soft-label approach to conduct AL for GNNs. ALLIE [5] designs AL specifically for large-scale imbalanced graph data. BIGENE [36] proposes a multi-agent Q-network consisting of a GCN module and a gated recurrent unit module for data selection.

Graph Structure Learning (GSL). GSL aims to learn both a graph learning model and a graph structure simultaneously. GCN-Jaccard [27] remove edges according to the Jaccard similarity of node features. RGCN [37] reduces the impacts of adversarial attacks by introducing variance-based attention weight in the message-passing. Pro-GNN [15] 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 [25] optimizes a graph structure model and an observation model to gain the optimal graph in an iterative manner.

8 Conclusion

Current GAL methods 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. After identifying the challenges of performing active learning on noisy graphs, we propose a novel iterative graph active learning framework by performing data selection and graph learning simultaneously. Not only high-quality data can be selected in `GALClean+`, but a cleaned graph will also be generated and utilized in the downstream graph tasks. Noticably, `GALClean+` enjoys a solid theoretical interpretation as an EM algorithm. Extensive experiments show that `GALClean+` has strong performance superiority over various baselines, especially when the existing noise is heavy.

9 Acknowledgement

This research is supported by the National Science Foundation (NSF) under grant numbers IIS-1909702, IIS-2153326 and IIS-2212145 and Army Research Office (ARO) under grant number W911NF21-1-0198.

References

[1] S. Balakrishnan, M. J. Wainwright, and B. Yu. Statistical guarantees for the em algorithm: From population to sample-based analysis. 2017.

[2] M. J. Beal. *Variational algorithms for approximate Bayesian inference*. University College London, 2003.

[3] H. Cai, V. W. Zheng, and K. C.-C. Chang. Active learning for graph embedding. *arXiv preprint arXiv:1705.05085*, 2017.

[4] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. In *ICML*. PMLR, 2020.

[5] L. Cui, X. Tang, S. Katariya, N. Rao, P. Agrawal, K. Subbian, and D. Lee. Allie:active learning on large-scale imbalanced graphs. In *Proceedings of the ACM Web Conference*, 2022.

[6] E. Dai, W. Jin, H. Liu, and S. Wang. Towards robust graph neural networks for noisy graphs with sparse labels. In *Proceedings of the Fifteenth ACM WSDM*, 2022.

[7] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society: series B*.

[8] F. Diehl, T. Brunner, M. T. Le, and A. Knoll. Graph neural networks for modelling traffic participant interaction. In *IEEE Intelligent Vehicles Symposium*, 2019.

[9] C. Gao, X. Wang, X. He, and Y. Li. Graph neural networks for recommender system. In *Proceedings of the Fifteenth ACM WSDM*, pages 1623–1625, 2022.

[10] L. Gao, H. Yang, C. Zhou, J. Wu, S. Pan, and Y. Hu. Active discriminative network representation learning. In *IJCAI*, 2018.

[11] J. Gasteiger, A. Bojchevski, and S. Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.

[12] S. Hu, Z. Xiong, M. Qu, X. Yuan, M.-A. Côté, Z. Liu, and J. Tang. Graph policy network for transferable active learning on graphs. *NeurIPS*, 33:10174–10185, 2020.

[13] Y. Hu, H. You, Z. Wang, Z. Wang, E. Zhou, and Y. Gao. Graph-mlp: node classification without message passing in graph. *arXiv preprint arXiv:2106.04051*, 2021.

[14] T. Huang, Y. Dong, M. Ding, Z. Yang, W. Feng, X. Wang, and J. Tang. Mixgcf: An improved training method for graph neural network-based recommender systems. In *Proceedings of the 27th ACM SIGKDD*, pages 665–674, 2021.

[15] W. Jin, Y. Ma, X. Liu, X. Tang, S. Wang, and J. Tang. Graph structure learning for robust graph neural networks. In *Proceedings of the 26th ACM SIGKDD*, pages 66–74, 2020.

[16] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan. Supervised contrastive learning. *NeurIPS*, 33, 2020.

[17] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[18] J. Liu, Y. Wang, B. Hooi, R. Yang, and X. Xiao. Lscale: Latent space clustering-based active learning. 2020.

[19] Y. Liu, X. Ao, Z. Qin, J. Chi, J. Feng, H. Yang, and Q. He. a gnn-based imbalanced learning approach for fraud detection. In *Proceedings of the Web Conference*, 2021.

[20] K. Madhawa and T. Murata. Active learning for node classification: an evaluation. *Entropy*, 22(10):1164, 2020.

[21] A. v. d. Oord, Y. Li, and O. Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[22] G. Papandreou, L.-C. Chen, K. P. Murphy, and A. L. Yuille. Weakly-and semi-supervised learning of a deep convolutional network for semantic image segmentation. In *Proceedings of the IEEE international conference on computer vision*.

[23] O. Sener and S. Savarese. Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489*, 2017.

[24] D. J. Wang, X. Shi, D. A. McFarland, and J. Leskovec. Measurement error in network data. *Social Networks*.

[25] R. Wang, S. Mou, X. Wang, W. Xiao, Q. Ju, C. Shi, and X. Xie. Graph structure estimation neural networks. In *Proceedings of the Web Conference 2021*, pages 342–353, 2021.

[26] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger. Simplifying graph convolutional networks. In *ICML*. PMLR, 2019.

[27] H. Wu, C. Wang, Y. Tyshetskiy, A. Docherty, K. Lu, and L. Zhu. Adversarial examples on graph data. *arXiv preprint arXiv:1903.01610*, 2019.

[28] Y. Wu, Y. Xu, A. Singh, Y. Yang, and A. Dubrawski. Active learning for graph neural networks via node feature propagation. *arXiv preprint arXiv:1910.07567*, 2019.

[29] Y. Xie, Y. Xiong, and Y. Zhu. Sast-gnn: a self-attention based spatio-temporal graph neural network for traffic prediction. In *International Conference on Database Systems for Advanced Applications*, pages 707–714. Springer, 2020.

[30] Y. Zeng and J. Tang. Rlc-gnn: An improved deep architecture for spatial-based graph neural network with application to fraud detection. *Applied Sciences*, 2021.

[31] W. Zhang, Y. Shen, Y. Li, L. Chen, Z. Yang, and B. Cui. Alg: fast and accurate active learning framework for graph convolutional networks. In *Proceedings of the 2021 SIGMOD*, pages 2366–2374, 2021.

[32] W. Zhang, Y. Wang, Z. You, M. Cao, P. Huang, J. Shan, Z. Yang, and B. Cui. Rim: Reliable influence-based active learning on graphs. *NeurIPS*, 34, 2021.

[33] W. Zhang, Y. Wang, Z. You, M. Cao, P. Huang, J. Shan, Z. Yang, and B. Cui. Information gain propagation: a new way to graph active learning with soft labels. *arXiv preprint arXiv:2203.01093*, 2022.

[34] W. Zhang, Z. Yang, Y. Wang, Y. Shen, Y. Li, L. Wang, and B. Cui. Grain: Improving data efficiency of graph neural networks via diversified influence maximization. *arXiv preprint arXiv:2108.00219*, 2021.

[35] Y. Zhang, M. Lease, and B. Wallace. Active discriminative text representation learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017.

[36] Y. Zhang, H. Tong, Y. Xia, Y. Zhu, Y. Chi, and L. Ying. Batch active learning with graph neural networks via multi-agent deep reinforcement learning. In *Proceedings of AAAI*, 2022.

[37] D. Zhu, Z. Zhang, P. Cui, and W. Zhu. Robust graph convolutional networks against adversarial attacks. In *Proceedings of the 25th ACM SIGKDD*, 2019.

[38] R. Zhu, L. Wang, C. Zhai, and Q. Gu. High-dimensional variance-reduced stochastic gradient expectation-maximization algorithm. In *ICML*. PMLR, 2017.

[39] Y. Zhu, W. Xu, J. Zhang, Y. Du, J. Zhang, Q. Liu, C. Yang, and S. Wu. A survey on graph structure learning: Progress and opportunities. *arXiv e-prints*, pages arXiv–2103, 2021.