Reliable Representations Make A Stronger Defender: Unsupervised Structure Refinement for Robust GNN

Kuan Li[†]
Institute of Computing Technology, Chinese Academy of Sciences
University of Chinese Academy of Sciences, Beijing, China
likuan_buaa@163.com

Yang Liu[†]
Institute of Computing Technology, Chinese Academy of Sciences
University of Chinese Academy of Sciences, Beijing, China
liuyang520ict@gmail.com

Xiang Ao[*†]
Institute of Computing Technology, Chinese Academy of Sciences
University of Chinese Academy of Sciences, Beijing, China
aoxiang@ict.ac.cn

Jianfeng Chi
Jinghua Feng
Hao Yang
Alibaba Group, Hangzhou, China
jinghua.fengjh@alibaba-inc.com
youhiroshi.yangh@alibaba-inc.com
bianfu.cjf@alibaba-inc.com

Qing He[†]
Institute of Computing Technology, Chinese Academy of Sciences
University of Chinese Academy of Sciences, Beijing, China
heqing@ict.ac.cn

ABSTRACT

Benefiting from the message passing mechanism, Graph Neural Networks (GNNs) have been successful on flourish tasks over graph data. However, recent studies have shown that attackers can catastrophically degrade the performance of GNNs by maliciously modifying the graph structure. A straightforward solution to remedy this issue is to model the edge weights by learning a metric function between pairwise representations of two end nodes, which attempts to assign low weights to adversarial edges. The existing methods use either raw features or representations learned by supervised GNNs to model the edge weights. However, both strategies are faced with some immediate problems: raw features cannot represent various properties of nodes (*e.g.*, structure information), and representations learned by supervised GNN may suffer from the poor performance of the classifier on the poisoned graph. We need representations that carry both feature information and as mush correct structure information as possible and are insensitive to structural perturbations. To this end, we propose an unsupervised pipeline, named STABLE, to optimize the graph structure. Finally, we input the well-refined graph into a downstream classifier. For this part, we design an advanced GCN that significantly enhances the robustness of vanilla GCN [24] without increasing the time complexity. Extensive experiments on four real-world graph benchmarks demonstrate that STABLE outperforms the state-of-the-art methods and successfully defends against various attacks.

[*]Corresponding author.
[†]Key Lab of Intelligent Information Processing of Chinese Academy of Sciences (CAS). Xiang Ao is also at Institute of Intelligent Computing Technology, Suzhou, China.

KDD '22, August 14–18, 2022, Washington, DC, USA

ACM ISBN 978-1-4503-9385-0/22/08.
https://doi.org/10.1145/3534678.3539484

CCS CONCEPTS

• **Mathematics of computing** → *Graph algorithms*; • **Security and privacy** → *Social network security and privacy*.

KEYWORDS

Graph Neural Network, Graph Adversarial Attack, Structure Learning

ACM Reference Format:
Kuan Li, Yang Liu, Xiang Ao, Jianfeng Chi, Jinghua Feng, Hao Yang, and Qing He. 2022. Reliable Representations Make A Stronger Defender: Unsupervised Structure Refinement for Robust GNN. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3534678.3539484

1 INTRODUCTION

Graphs are ubiquitous data structures that can represent various objects and their complex relations [2, 29, 46, 48, 50]. As a powerful tool of learning representations from graphs, Graph Neural Networks (GNNs) burgeon widely explorations in recent years [14, 24, 27, 35, 42] for numerous graph-based tasks, primarily focused on node representation learning [23, 33, 36, 43] and transductive node classification [14, 18, 24, 35]. The key to the success of GNNs is the neural message passing mechanism, in which GNNs regard features and hidden representations as messages carried by nodes and propagate them through the edges. However, this mechanism also brings a security risk.

Recent studies have shown that GNNs are vulnerable to adversarial attacks [7, 40, 51, 54, 55]. In other words, by limitedly rewiring the graph structure or just perturbing a small part of the node features, attackers can easily fool the GNNs to misclassify nodes in the graph. The robustness of the model is essential for some security-critical domains. For instance, in fraudulent transaction detection, fraudsters can conceal themselves by deliberately dealing

Table 1: The mean accuracy under different perturbation rates by MetaAttack on the Cora dataset. Here the perturbation rate is the ratio of changed edges. The data split follows 10%/ 10%/ 80/%(train/ validation/ test).

Ptb Rate	GCN	GRCN	GNNGuard	Jaccard
0%	83.56	**86.12**	78.52	81.79
5%	76.36	**80.78**	77.96	80.23
10%	71.62	72.42	**74.86**	74.65
20%	60.31	65.43	72.03	**73.11**

with common users. Thus, it is necessary to study the robustness of GNNs. Although attackers can modify the clean graph by perturbing node features or the graph structure, most of the existing adversarial attacks on graph data have concentrated on modifying graph structure [10, 55]. Moreover, the structure perturbation is considered more effective [40, 54] probably due to the message passing mechanism. Misleading messages will pass through a newly added edge, or correct messages cannot be propagated because an original edge is deleted. In this paper, our purpose is to defend against the non-targeted adversarial attack on graph data that attempts to reduce the overall performance of the GNN. Under this setting, the GNNs are supposed to be trained on a graph that the structure has already been perturbed.

One representative perspective to defend against the attack is to refine the graph structure by reweighting the edges [52]. Specifically, edge weights are derived from learning a metric function between pairwise representations [9, 25, 35, 37, 45]. Intuitively, the weight of an edge could be represented as a distance measure between two end nodes, and defenders can further prune or add edges according to such distances.

Though extensive approaches have been proposed to model the pairwise weights, most research efforts are devoted to designing a novel metric function, while the rationality of the inputs of the function is inadequately discussed. In more detail, they usually utilize the original features or representations learned by the supervised GNNs to compute the weights.

However, optimizing graph structures based on either features or supervised signals might not be reliable. For example, GNNGuard [45] and Jaccard[40] utilize cosine similarity of the initial features to model the edge weights, while GRCN [44] uses the inner product of learned representations. The performance of these three models on Cora attacked by MetaAttack [55][1] is listed in Table 1. From the table, we first observe feature-based methods do not perform well under low perturbation rates, because features cannot carry structural information. Optimizing the structure based on such an insufficient property can lead to mistakenly deleting normal edges (the statistics of the removed edges is listed in Table 3). When the perturbation is low, the negative impact of such misdeletion is greater than the positive impact of deleting malicious edges.Thus, we want to refine the structure by learned representations which contain structural information. Second, we also see the representations learned by the supervised GNNs are not reliable under high perturbations (the results of GRCN). This is probably because attack methods are designed to degrade the accuracy of a surrogate GNN, so the quality of representations learned by the classifier co-vary with the task performance.

Based on the above analysis, we consider that the representations used for structure refining should be obtained in a different manner, and two factors in terms of learning representations in the adversarial scenario should be highlighted: 1) **carrying feature information and in the meantime carrying as much correct structure information as possible** and 2) **insensitivity to structural perturbations**.

To this end, we propose an approach named STABLE (STructure leArning GNN via more reliaBLe rEpresentations) in this paper, and it learns the representations used for structure refining by unsupervised learning. The unsupervised approach is relatively reliable because the objective is not directly attacked. Additionally, the unsupervised pipeline can be viewed as a kind of pretraining, and the learned representations may have been trained to be invariant to certain useful properties [39] (i.e., the perturbed structure here). We design a contrastive method with a novel pre-process and recovery schema to obtain the representations. Different from the previous contrastive method [15, 32, 36, 53], we roughly refine the graph to remove the easily detectable adversarial edges and generate augmentation views by randomly recovering a small portion of removed edges. Pre-processing makes the underlying structural information obtained during the representation learning process relatively correct, and such an augmentation strategy can be viewed as injecting slight attack to the pre-processed graph. Then the representations learned on different augmentation views tend to be similar during the contrastive training. That is to say, we obtain representations insensitive to the various slight attacks. Such learned representations fulfill our requirements and can be utilized to perform graph structure refinement to derive an unpolluted graph.

In addition, any GNNs can be used for the downstream learning tasks after the structure is well-refined. For this part, many methods [20, 22] just use the vanilla GCN [24]. By observing what makes edge insertion or deletion a strong adversarial change, we find that GCN falls victim to its renormalization trick. Hence we introduce an advanced message passing in the GCN module to further improve the robustness.

Our contributions can be summarized as follow:

(1) We propose a contrastive method with robustness-oriented augmentations to obtain the representations used for structure refining, which can effectively capture structural information of nodes and are insensitive to the perturbations. (2) We further explore the reason for the lack of robustness of GCN and propose a more robust normalization trick. (3) Extensive experiments on four real-world datasets demonstrate that STABLE can defend against different types of adversarial attacks and outperform the state-of-the-art defense models.

2 PRELIMINARIES

2.1 Graph Neural Networks

Let $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathbf{X}\}$ represent a graph with N nodes, where $\mathcal{V}$ is the node set, $\mathcal{E}$ is the edge set, and $\mathbf{X} \in \mathbb{R}^{N \times d}$ is the original

[1] It is the state-of-the-art attack method that uses meta-gradients to maliciously modify the graph structure.

feature matrix of nodes. Let $\mathbf{A} \in \{0,1\}^{N\times N}$ represent the adjacency matrix of $\mathcal{G}$, in which $\mathbf{A}_{ij} \in \{0,1\}$ denotes the existence of the edge $e_{ij} \in \mathcal{E}$ that links node v_i and v_j. The first-order neighborhood of node v_i is denoted as $\mathcal{N}_i$, including node v_i itself. We use $\mathcal{N}_i^*$ to indicate v_i's neighborhood excluding itself. The transductive node classification task can be formulated as we now describe. Given a graph $\mathcal{G}$ and a subset $\mathcal{V}_L \subseteq \mathcal{V}$ of labeled nodes, with class labels from $\mathcal{C} = \{c_1, c_2, ..., c_K\}$. $\mathcal{Y}_L$ and $\mathcal{Y}_U$ denote the ground-truth labels of labeled nodes and unlabeled nodes, respectively. The goal is to train a GNN f_θ to learn a function: $\mathcal{V} \rightarrow \mathcal{Y}$ that maps the nodes to the label set so that f_θ can predict labels of unlabeled nodes. θ is the trainable parameters of GNN.

GNNs can be generally specified as $f_\theta(\mathbf{X}, \mathbf{A})$ [14, 24, 48]. Each layer can be divided into a message passing function (MSP) and an updating function (UPD). Given a node v_i and its neighborhood $\mathcal{N}_i^*$, GNN first implements MSP to aggregate information from $\mathcal{N}_i^*$: $\boldsymbol{m}_i^t = \text{MSP}(\{\boldsymbol{h}_j^{t-1}; j \in \mathcal{N}_i^*\})$, where $\boldsymbol{h}_j^{t-1}$ denotes the hidden representation in the previous layer, and $\boldsymbol{h}_j^0 = \boldsymbol{x}_j$. Then, GNN updates the representation by UPD: $\boldsymbol{h}_i^t = \text{UPD}(\boldsymbol{m}_i^t, \boldsymbol{h}_i^{t-1})$, which is usually a sum or concat function. GNNs can be designed in an end-to-end fashion and can also serve as a representation learner for downstream tasks [24].

2.2 Gray-box Poisoning Attacks and Defence

In this paper, we explore the robustness of GNNs under non-targeted Gray-box poisoning attack. Gray-box [44] means the attacker holds the same data information as the defender, and the defense model is unknown. Poisoning attack represents GNNs are trained on a graph that attackers maliciously modify, and the aim of it is to find an optimal perturbed $\mathbf{A}'$, which can be formulated as a bilevel optimization problem [54, 55]:

$$\begin{aligned} &\underset{A' \in \Phi(A)}{\text{argmin}} \quad \mathcal{L}_{atk}(f_{\theta^*}(\mathbf{A}', \mathbf{X})) \\ &s.t. \quad \theta^* = \underset{\theta}{\text{argmin}}\, \mathcal{L}_{train}(f_\theta(\mathbf{A}', \mathbf{X})). \end{aligned} \tag{1}$$

Here $\Phi(\mathbf{A})$ is a set of adjacency matrix that fit the constraint: $\frac{\|\mathbf{A}'-\mathbf{A}\|_0}{\|\mathbf{A}\|_0} \leq \Delta$, $\mathcal{L}_{atk}$ is the attack loss function, and $\mathcal{L}_{train}$ is the training loss of GNN. The θ^* is the optimal parameter for f_θ on the perturbed graph. Δ is the maximum perturbation rate. In the non-targeted attack setting, the attacker aims to degrade the overall performance of the classifier.

Many efforts have been made to improve the robustness of GNNs [8, 20, 22, 28, 41, 49]. Among them, metric learning approach [52] is a promising method [5, 25, 37, 45], which refines the graph structure by learning a pairwise metric function $\phi(\cdot,\cdot)$:

$$\mathbf{S}_{ij} = \phi(\boldsymbol{z}_i, \boldsymbol{z}_j), \tag{2}$$

where $\boldsymbol{z}_i$ is the raw feature or the learned embedding of node v_i produced by GNN, and $\mathbf{S}_{ij}$ denotes the learned edge weight between node v_i and v_j. $\phi(\cdot,\cdot)$ can be a similarity metric or implemented in multilayer perceptions (MLPs). Further, the structure can be optimized according to the weights matrix $\mathbf{S}$. For example, GNN-Guard [45] utilizes cosine similarity to model the edge weights and then calculates edge pruning probability through a non-linear transformation. Moreover, GRCN [44] and GAUGM [47] directly compute the weights of the edges by the inner product of embeddings, and no additional parameters are needed:

$$\mathbf{S}_{ij} = \sigma(\mathbf{Z}\mathbf{Z}^{\mathrm{T}}). \tag{3}$$

Such methods aim to assign high weights to helpful edges and low weights to adversarial edges or even remove them. Here we want to define what is a helpful edge. Existing literature posits that strong homophily of the underlying graph is necessary for GNNs to achieve good performance on transductive node classification [1, 6, 17]. Under the homophily assumption [31], the edge which links two similar nodes (*e.g.*, same label) might help the nodes to be correctly classified. However, since only few labeled nodes are available, previous works utilize the raw features or the representations learned by the task-relevant GNNs to search for similar nodes.

Different from the aforementioned defense methods, we leverage an unsupervised pipeline to calculate the weights matrix and refine the graph structure.

3 METHOD

In this section, we will present STABLE in a top-down fashion: starting with an elaboration of the overall framework, followed by the specific details of the structural optimization module, and concluding by an exposition of the advanced GCN.

3.1 The Overall Architecture

The illustration of the framework is shown in Fig. 1. It consists of two major components, the structure learning network and the advanced GCN classifier. The structure learning network can be further divided into two parts, representation learning and graph refining. For the representation learning part, we design a novel contrastive learning method with a robustness-oriented graph data augmentation, which randomly recovers the roughly removed edges. Then the learned representations are utilized to revise the structure according to the homophily assumption [31]. Finally, in the classification step, by observing what properties the nodes connected by the adversarial edges have, we carefully change the renormalization trick in GCN[24] to improve the robustness. The overall training algorithm is shown in Appendix A.1.

3.2 Representation learning

According to [40, 55], most of the perturbations are edge insertions, which connect nodes with different labels. Thus a straightforward method to deal with the structural perturbation is to find the adversarial edges and remove them. Previous works utilize the raw features or the representations learned by the end-to-end GNNs to seek the potential perturbations. As we mentioned before, such approaches have flaws.

STABLE avoids the pitfalls by leveraging a task-irrelevant contrastive method with robustness-oriented augmentations to learn the representations. First, we roughly pre-process the perturbed graph based on a similarity measure, *e.g.*, Jaccard similarity or cosine similarity. We quantify the score $\mathbf{S}_{ij}$ between node v_i and its neighbor v_j as follows:

$$\mathbf{S}_{ij} = \text{sim}(\boldsymbol{x}_i, \boldsymbol{x}_j), \tag{4}$$

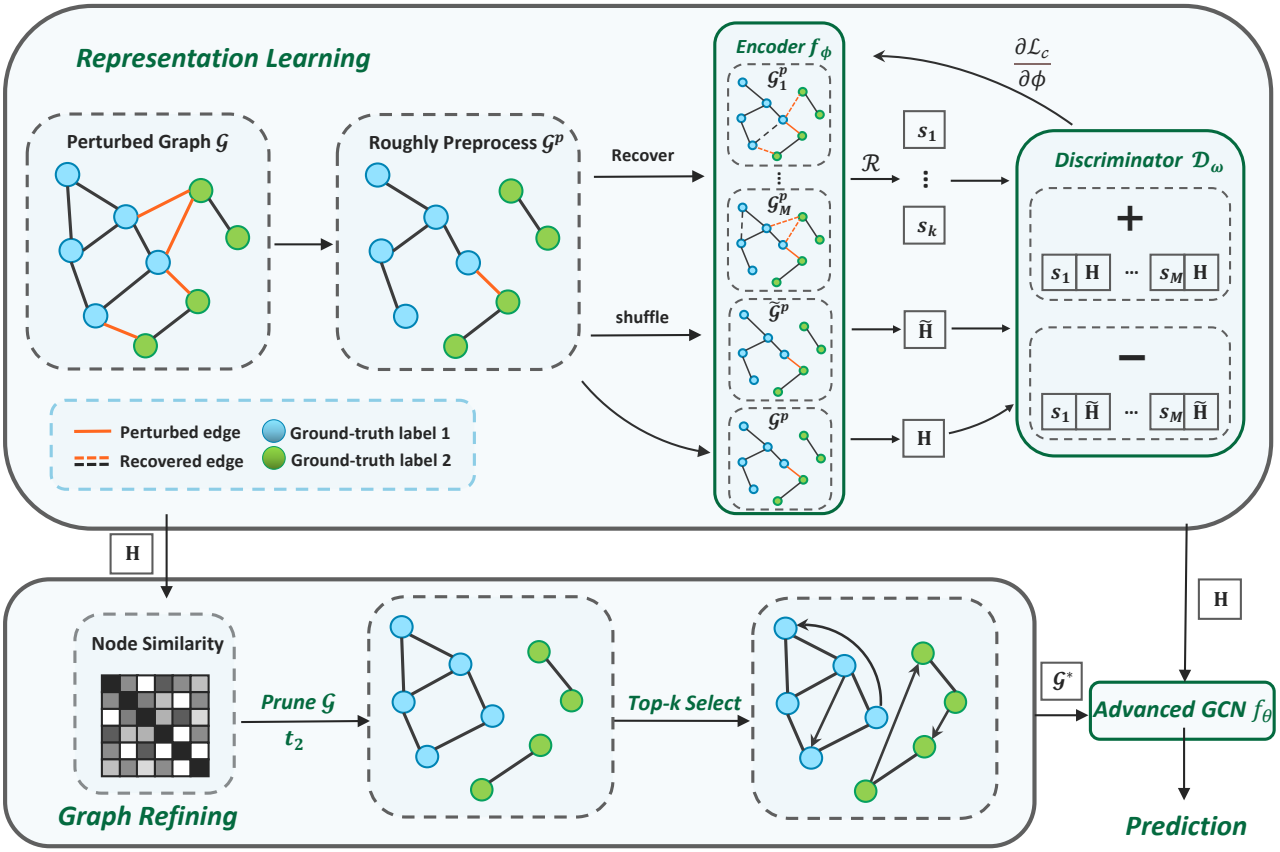


Figure 1: Overall framework. Dash lines mean recovered edges.

where $\mathcal{S}$ is the score matrix, $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are the feature vectors of v_i and v_j, respectively. Then we prune the edges whose scores are below a threshold t_1. The removed edge matrix and roughly pre-processed graph are denoted as $\mathbf{E}$ and $\mathcal{G}^P$, where $\mathbf{E}_{ij} = 1$ denotes the edge between v_i and v_j is removed.

Generating views is the key component of contrastive methods. Different views of a graph provide different contexts for each node, and we hope our views carry the context for enhancing the robustness of the representations. Previous works generate augmentations by randomly perturbing the structure [43, 53] on the initial graph. Different from them, we generate M graph views, denoted as $\mathcal{G}_1^P$ to $\mathcal{G}_M^P$, by randomly recovering a small portion of $\mathbf{E}$ on the pre-processed graph $\mathcal{G}^P$. Formally, we first sample a random masking matrix $\mathbf{P} \in \{0,1\}^{N\times N}$, whose entry is drawn from a Bernoulli distribution $\mathbf{P}_{ij} \sim \mathcal{B}(1-p)$ if $\mathbf{E}_{ij} = 1$ and $\mathbf{P}_{ij} = 0$ otherwise. Here p is a hyper-parameter that controls the recovery ratio. The adjacency matrix of the augmentation view can be computed as:

$$\mathsf{A}_1^P = \mathsf{A}^P + \mathbf{E} \odot \mathbf{P}, \tag{5}$$

where A^P and A_1^P denote the adjacency matrix of $\mathcal{G}^P$ and $\mathcal{G}_1^P$ respectively, and $\odot$ is the Hadamard product. Other views can be obtained in the same way. The pre-process and the robustness-oriented augmentations are critical to the robustness, and we will elaborate on this point at the end of this subsection.

Our objective is to learn a one-layer GCN encoder f_ϕ parameterized by ϕ to maximize the mutual information between the local representation in $\mathcal{G}^P$ and the global representations in $\mathcal{G}_j$. Here $\mathcal{G}_j$ denote arbitrary augmentation j. The encoder follows the propagation rule:

$$f_\phi(\mathsf{X}, \mathsf{A}) = \sigma(\hat{\mathsf{A}}\mathsf{X}\mathbf{W}_\phi) \tag{6}$$

where $\hat{\mathsf{A}} = (\mathbf{D}+\mathbf{I}_N)^{-\frac{1}{2}}(\mathsf{A}+\mathbf{I}_N)(\mathbf{D}+\mathbf{I}_N)^{-\frac{1}{2}}$ is the renormalized adjacency matrix. we denote node representations in $\mathcal{G}^P$ and augmentation j as $\mathbf{H} = f_\phi(\mathsf{X}, \mathsf{A}^P)$ and $\mathbf{H}_j = f_\phi(\mathsf{X}, \mathsf{A}_j^P)$. The global representation can be obtained via a readout function $\mathcal{R}$. We leverage a global average pooling function:

$$\boldsymbol{s}_j = \sigma\left(\frac{1}{N}\sum_{i=1}^{\tilde{\mathbf{N}}} \boldsymbol{h}_{i,j}\right), \tag{7}$$

where σ is the logistic sigmoid nonlinearity, and $\boldsymbol{h}_{i,j}$ is the representation of node v_i in $\mathcal{G}_j^P$. We employ a discriminator $\mathcal{D}_\omega$ to distinguish positive samples and negative samples, and the output $\mathcal{D}_\omega(\boldsymbol{h}, \boldsymbol{s})$ represents the probability score of the sample pair from the joint distribution. ω is the parameters of the discriminator. To generate negative samples, which means the pair sampled from the product of marginals, we randomly shuffle the nodes' features in $\mathcal{G}^P$. We denote the shuffled graph and its representation as $\widetilde{\mathcal{G}}^P$ and

$\mathbf{H} = f_\phi(\mathbf{X}, \mathbf{A}^P)$. To maximize the mutual information, we use a binary cross-entropy loss between the positive samples and negative samples, and the objective can be writen as:

$$\mathcal{L}_C = -\frac{1}{2N}\sum_{i=1}^{\tilde{N}}\frac{1}{k}\sum_{j=1}^{\tilde{N}} \log\mathcal{D}_\omega(\boldsymbol{h}_i, \boldsymbol{s}_1) \ldots + \log\mathcal{D}_\omega(\boldsymbol{h}_i, \boldsymbol{s}_M) + \log\ 1 - \mathcal{D}_\omega(\tilde{\boldsymbol{h}}_i, \boldsymbol{s}_1) \ \ldots + \log\ 1 - \mathcal{D}_\omega(\tilde{\boldsymbol{h}}_i, \boldsymbol{s}_M) \quad (8)$$

where $\boldsymbol{h}_i$ and $\tilde{\boldsymbol{h}}_i$ are the representations of node v_i in $\mathcal{G}^P$ and $\tilde{\mathcal{G}}^P$. Minimizing this objective function can effectively maximize the mutual information between the local representations in $\mathcal{G}^P$ and the global representations in the augmentations based on the Jensen-Shannon MI (mutual information) estimator[16].

To clarify why pre-processing and the designed augmentation method are crucial to robustness, we count the results of the pre-process and the recovery: Using Jaccard as the measure of the rough pre-process, 1,705 edges are removed on Cora under 20% perturbation rate. Among them, 691 are adversarial edges, and 1014 are normal edges. 259 edges out of 1,014 connect two nodes with different labels. More than half of the removals are helpful. Therefore, most of the edges in $\mathbf{E}$ can be regarded as adversarial edges so that **the recovery can be viewed as injecting slight attacks on** $\mathcal{G}^P$. The degrees of perturbation can be ranked as $\mathcal{G} \gg \mathcal{G}_1^P \approx \mathcal{G}_2^P \ldots \approx \mathcal{G}_M^P > \mathcal{G}^P$.

Recall that we need representations that carry as much correct structural information as possible and are insensitive to perturbations. We train our encoder on the three much cleaner graphs, so the representations are much better than those learned in the original graph. The process of maximizing mutual information makes the representations learned in $\mathcal{G}^P$ and in the views similar so that the representations will be insensitive to the recovery. In other words, they will be insensitive to perturbations. In short, the rough pre-process meets the former requirement, and the robustness-oriented augmentation meets the latter.

3.3 Graph Refining

Once the high-quality representations are prepared, we can further refine the graph structure in this section. Existing literature posits that strong homophily of the underlying graph can help GNNs to achieve good performance on transductive node classification [1, 6, 17]. Therefore, the goal of the refining is to reduce heterophilic edges (connecting two nodes in the different classes) and add homophilic edges (connecting two nodes in the same class). Under homophily assumption [31], similar nodes are more likely in the same class, so we use node similarity to search for potential homophilic edges and heterophilic edges.

We leverage the representations learned by the contrastive learning encoder to measure the nodes similarity:

$$\mathbf{M}_{ij} = \mathrm{sim}(\boldsymbol{h}_i, \boldsymbol{h}_j), \quad (9)$$

where $\mathbf{M}$ is the similarity score matrix, and $\mathbf{M}_{ij}$ is the score of node v_i and node v_j. Here we utilize the cosine similarity. Then we remove all the edges that connect nodes with similarity scores below a threshold t_2. The above process can be formulated as:

$$\mathbf{A}^R_{ij} = \begin{cases} 1 & if\ \mathbf{M}_{ij} > t_2\ and\ \mathbf{A}_{ij} = 1 \\ 0 & otherwise, \end{cases} \quad (10)$$

where $\mathbf{A}^R$ is the adjacency matrix after pruning.

After pruning, we insert some helpful edges by building a top-k matrix $\mathbf{T}^k$. For each node, we connect it to k nodes that are most similar to it. Formally, $\mathbf{T}^k_{ij} = 1$, if v_j is one of the k nodes most similar to v_i. Note that $\mathbf{T}^k$ is not a symmetric matrix because the similarity relation is not symmetric. For example, v_i is the most similar node to v_j but not vise versa. The pruning strategy cannot eliminate all the perturbations, and these insertions can effectively diminish the impact of the remaining harmful edges. From our empirical results, this is particularly effective when the perturbation rate is high. The optimal adjacency matrix can be formulated as:

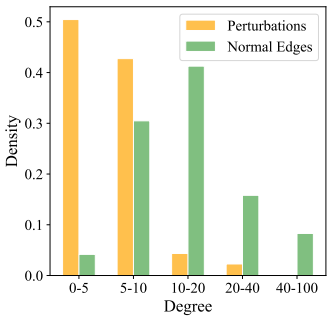
$$\mathbf{A}^* = \mathbf{A}^R + \mathbf{T}^k \quad (11)$$

The optimal graph is denoted as $\mathcal{G}^*$, and it is a directed graph due to the asymmetry of $\mathbf{T}$.

3.4 Advanced GCN In the Adversarial Attack Scenario

After the graph is well-refined, any GNNs can be used for the downstream task. We find that only one modification to the renormalization trick in GCN is needed to enhance the robustness greatly. This improvement does not introduce additional learnable parameters and therefore does not reduce the efficiency of the GCN.

[55] studied the node degree distributions of the clean graph and the node degrees of the nodes that are picked for adversarial edges. Not surprisingly, the nodes selected by the adversarial edges are those with low degrees. The nodes with only a few neighbors are more vulnerable than high-degree nodes due to the message passing mechanism. However, the discussion about the picked nodes only exposes one-side property of the adversarial edges. We further define the degree of an edge as the sum of both connected nodes' degrees. The edge degree distribution is shown in Fig. 2 (a). It shows that adversarial edges tend to link **two** low-degree nodes, and nearly half of them are below 5 degrees.



(a)

Δ	GCN	GCN*
0%	**83.56**	82.76
5%	76.36	**78.17**
10%	71.62	**74.23**
20%	60.31	**69.59**

(b)

Figure 2: (a): The edge degree distribution of the clean graph and the distribution of the adversarial edges on Cora attacked by MetaAttack. (b): The mean accuracy under different perturbation rates on Cora. The data split follows 10%/ 10%/ 80/%(train/ val/test).

GCN assigns higher aggregation weights to low-degree neighbors, and the renormalization adjacency matrix is formulated as: $\hat{\mathbf{A}} = (\mathbf{D}+\mathbf{I}_N)^{-\frac{1}{2}}(\mathbf{A}+\mathbf{I}_N)(\mathbf{D}+\mathbf{I}_N)^{-\frac{1}{2}}$. However, based on the above analysis, low-degree neighbors are more likely to be maliciously added neighbors. To give a specific example, here is node v_i with only one neighbor in the clean graph, and the attacker link it to a low-degree node. In the aggregation step, only half of the messages are are correct. Even worse, the misleading message is assigned a higher weight because the added neighbor has a lower degree than the original neighbor.

Hence, a simple but effective way to improve the robustness of GCN is to assign lower weights to the low-degree neighbors. We propose our advanced GCN, which will implement the message passing as follow:

$$\boldsymbol{h}_i^t = \text{ReLU}\left(\tilde{\sum_{j\in\mathcal{N}_i^*}} \frac{(d_i d_j)^\alpha}{Z}\boldsymbol{h}_j^{t-1} + \beta \boldsymbol{h}_i^{(t-1)}\ \mathbf{W}_\theta^t\right) \tag{12}$$

where α is the hyper-parameter that controls the weight design scheme, β is the hyper-parameter that controls the weight of self-loop, Z is a normalization coefficient, and $\text{ReLU}(\cdot) = max(0,\cdot)$. h_i^0 is the representation learned in Section 3.2. When α is large, a node aggregate more from the high-degree nodes. β is also designed for robustness. Once the neighbors are unreliable, nodes should refer more to themselves.

High-degree nodes can be viewed as robust nodes, because a few perturbations can hardly cancel out the helpful information brought by their original neighbors. The learnable attack algorithms tend to bypass them. Thus, the message from these nodes are usually more reliable.

We show an example in Fig. 2 (b). GCN* denotes the advanced GCN with $\alpha = 0.6$ and $\beta = 2$. As the perturbation rate increases, the robustness improves more obviously. In fact, this normalization trick can merge with any GNNs.

We train an advanced GCN f_θ by minimizing the cross-entropy:

$$\mathcal{L}_G = \tilde{\sum_{i\in\mathcal{V}_L}} \ell(f_\theta(\mathbf{H}, \mathbf{A}^*)_i, y_i) \tag{13}$$

where ℓ is the cross entropy loss.

4 EXPERIMENTS

In this section, we empirically analyze the robustness and effectiveness of STABLE against different types of adversarial attacks. Specifically, we aim to answer the following research questions:

- RQ1: Does STABLE outperform the state-of-the-art defense models under different types of adversarial attacks?
- RQ2: Is the structure learned by STABLE better than learned by other methods?
- RQ3: What is the performance with respect to different training parameters?
- RQ4: How do the key components benefit the robustness? (see Appendix A.5)

4.1 Experimental Settings

4.1.1 Datasets. Following [8, 22, 54], we conduct our experiments on four benchmark datasets, including three citation graphs, *i.e.*, Cora, Citeseer, and PubMed, and one blog graph, *i.e.*, Polblogs. The statistics of these datasets are shown in Appendix A.2.

4.1.2 Baselines. We compare STABLE with representative and state-of-the-art defence GNNs to verify the robustness. The baselines and SOTA approaches are **GCN** [24], **RGCN** [49], **Jaccard** [40], **GNNGuard** [45], **GRCN** [44], **ProGNN** [22], **SimpGCN** [20], **Elastic** [28]. We implement three non-targeted structural adversarial attack methods, *i.e.*, **MetaAttack [55]**, **DICE [38]** and **Random**.

We introduce all these defence methods and attack methods in the Appendix A.3.

4.1.3 Implementation Details. The implementation details and parameter settings are introduced in Appendix A.4

4.2 Robustness Evaluation (RQ1)

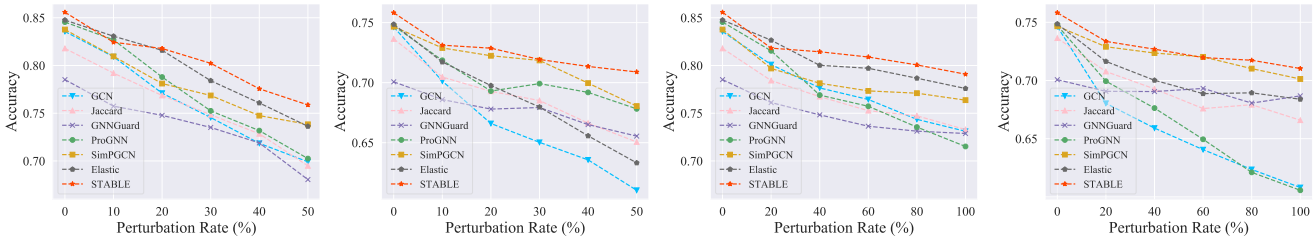
To answer RQ1, we evaluate the performance of all methods attacked by different methods.

4.2.1 Against Me ack. Table 2 shows the performance on three datasets against MetaAttack [55], which is an effective attack method. The top two performance is highlighted in bold and underline. We set the perturbation rate from 0% to 20%. From this main experiment, we have the following observations:

- All the methods perform closely on clean graphs because most of them are designed for adversarial attacks, and the performance on clean graphs is their goal. Both GCN and RGCN perform poorly on graphs with perturbations, proving that GNNs are vulnerable without valid defenses.
- Jaccard, GNNGuard, GRCN, and ProGNN are all structure learning methods. Jaccard and GNNGuard seem insensitive to the perturbations, but there is a trade-off between the performance and the robustness. They prune the edges based on the features, but the raw features cannot sufficiently represent nodes' various properties. ProGNN and GRCN perform well when the perturbation rate is low, and the accuracy drops rapidly as the perturbation rate rises. GRCN suffers from poor end-to-end representations, and for ProGNN, we guess it is hard to optimize the structure on a heavily contaminated graph directly. Compared with them, STABLE leverages the task-irrelevant representations to optimize the graph structure, which leads to higher performance.
- SimPGCN and Elastic are two recent state-of-the-art robust methods. SimPGCN is shown robust on Cora and Citeseer because it can adaptively balance structure and feature information. It performs poorly on Polblogs due to the lack of feature information, which can also prove that the robustness of SimpGCN relies on the raw features. Hence, it might also face the same problem that the original features miss some valuable information. Elastic benefits from its ability to model local smoothing. Different from them, our STABLE focus on refining the graph structure, and we just design a variant of GCN as the classifier. STABLE outperforms these two methods with 1%S8% improvement under low perturbation rate and 7%S24% improvement under large perturbation rate.
- STABLE outperforms other methods under different perturbation rates. Note that on Polblogs, we randomly remove and add a small portion of edges to generate augmentations. As the perturbation rate increases, the performance drops slowly on all datasets, which demonstrates that STABLE is insensitive to the perturbations.

Table 2: Classification accuracy(%) under different perturbation rates. The top two performance is highlighted in bold and underline.

Dataset	Ptb Rate	GCN	RGCN	Jaccard	GNNGuard	GRCN	ProGNN	SimPGCN	Elastic	STABLE
Cora	0%	83.56±0.25	83.85±0.32	81.79±0.37	78.52±0.46	**86.12±0.41**	84.55±0.30	83.77±0.57	84.76±0.53	85.58±0.56
	5%	76.36±0.84	76.54±0.49	80.23±0.74	77.96±0.54	80.78±0.94	79.84±0.49	78.98±1.10	**82.00±0.39**	81.40±0.54
	10%	71.62±1.22	72.11±0.99	74.65±1.48	74.86±0.54	72.43±0.78	74.22±0.31	75.07±2.09	76.18±0.46	**80.49±0.61**
	15%	66.37±1.97	65.52±1.12	74.29±1.11	74.15±1.64	70.72±1.13	72.75±0.74	71.42±3.29	74.41±0.97	**78.55±0.44**
	20%	60.31±1.98	63.23±0.93	73.11±0.88	72.03±1.11	65.34±1.24	64.40±0.59	68.90±3.22	69.64±0.62	**77.80±1.10**
Citeseer	0%	74.63±0.66	75.41±0.20	73.64±0.35	70.07±1.31	75.65±0.21	74.73±0.31	74.66±0.79	74.86±0.53	**75.82±0.41**
	5%	71.13±0.55	72.33±0.47	71.15±0.83	69.43±1.46	**74.47±0.38**	72.88±0.32	73.54±0.92	73.28±0.59	74.08±0.58
	10%	67.49±0.84	69.80±0.54	69.85±0.77	67.89±1.09	72.27±0.69	69.94±0.45	72.03±1.30	73.41±0.36	**73.45±0.40**
	15%	61.59±1.46	62.58±0.69	67.50±0.78	69.14±0.84	67.48±0.42	62.61±0.64	69.82±1.67	67.51±0.45	**73.15±0.53**
	20%	56.26±0.99	57.74±0.79	67.01±1.10	69.20±0.78	63.73±0.82	55.49±1.50	69.59±3.49	65.65±1.95	**72.76±0.53**
Polblogs	0%	95.04±0.11	95.38±0.14	/	/	94.89±0.24	95.93±0.17	94.86±0.46	95.57±0.26	**95.95±0.27**
	5%	77.55±0.77	76.46±0.47	/	/	80.37±0.46	93.48±0.54	75.08±1.08	90.08±1.06	**93.80±0.12**
	10%	70.40±1.13	70.35±0.40	/	/	69.72±1.36	85.81±1.00	68.36±1.88	84.05±1.94	**92.46±0.77**
	15%	68.49±0.49	67.74±0.50	/	/	66.56±0.93	75.60±0.70	65.02±0.74	72.17±0.74	**90.04±0.72**
	20%	68.47±0.54	67.31±0.24	/	/	68.20±0.71	73.66±0.64	64.78±1.33	71.76±0.92	**88.46±0.33**
Pubmed	0%	86.83±0.06	86.02±0.08	86.85±0.09	85.24±0.07	86.72±0.03	87.33±0.18	**88.12±0.17**	87.71±0.06	87.73± 0.11
	5%	83.18±0.06	82.37±0.12	86.22±0.08	84.65±0.09	84.85±0.07	87.25±0.09	86.96±0.18	86.82±0.13	**87.59±0.08**
	10%	81.24±0.17	80.12±0.12	85.64±0.08	84.51±0.06	81.77±0.13	87.25±0.09	86.41±0.34	86.78±0.11	**87.46±0.12**
	15%	78.63±0.10	77.33±0.16	84.57±0.11	84.78±0.10	77.32±0.13	87.20±0.09	85.98±0.30	86.36±0.14	**87.38±0.09**
	20%	77.08±0.2	74.96±0.23	83.67±0.08	84.25±0.07	69.89±0.21	87.09±0.10	85.62±0.40	86.04±0.17	**87.24±0.08**



(a) $\text{Cora}_{\text{DICE}}$ (b) $\text{Citeseer}_{\text{DICE}}$ (c) $\text{Cora}_{\text{Random}}$ (d) $\text{Citeseer}_{\text{Random}}$

Figure 3: Cora and Citeseer under DICE and Random

Especially STABLE has a huge improvement over other methods on heavily contaminated graphs.

4.2.2 Other attacks. The performance under DICE and Random is presented in Fig. 3. We only show the results on Cora and Citeseer due to the page limitation. RGCN and GRCN are not competitive, so they are not shown to keep the figure neat. Considering that DICE and Random are not as effective as MetaAttack, we set the perturbation rate higher. The figure shows that STABLE consistently outperforms all other baselines and successfully resists both attacks. Together with the observations from Section 4.2, we can conclude that STABLE outperforms these baselines and is able to defend against different types of attacks.

4.3 Result of Sturcture Learning (RQ2)

To validate the effectiveness of structure learning, we compare the results of structure optimization in STABLE and other metric learning methods. RGCN fails to refine the graph and makes the graph denser, so we exclude it. The statistics of the learned graphs are shown in Table 3. It shows the number of total removed edges, removed adversarial edges, and removed normal edges. Due to the limit of space, we only show results on Cora under MetaAttack.

To ensure a fair comparison, we tune the pruning thresholds in each method to close the number of total removed edges. It can be observed that STABLE achieves the highest pruning accuracy, indicating that STABLE revise the structure more precisely via more reliable representations.

4.4 Parameter Analysis (RQ3)

From our experimental experience, we mainly tune k and α to achieve peak performance. Thus, we alter their values to see how they affect the robustness of STABLE. The sensitivity of other parameters are presented in Appendix A.6. As illustrated in Fig. 4, we conduct experiments on Cora with the perturbation rate of 20% by

Table 3: The statistics of the removed edges on the learned graph.

Method	Total	Adversarial	Normal	Accuracy(%)
Jaccard	1,008	447	561	44.35
GNNGuard	1,082	482	600	44.55
STABLE	1,035	601	434	58.07

MetaAttack. It is worth noting that, regardless of the value of k, it is better to add than not to add. Another observation is that even $\alpha = 5$, which means nodes almost only aggregate messages from the neighbors with the highest degree, the result is still better than vanilla GCN, *i.e.*, $\alpha = -0.5$.

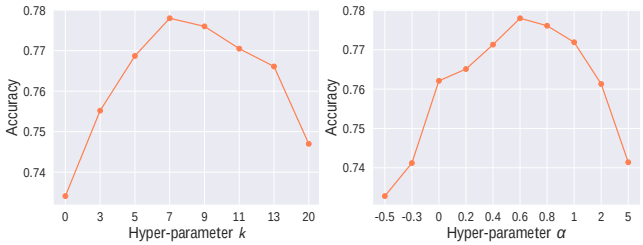


Figure 4: Parameter sensitivity analysis on Cora.

Moreover, to explore the relationship between these parameters and perturbation rates, we list the specific values which achieve the best performance on Cora in Table 4. Both k and α are directly proportional to the perturbation rate, which is consistent with our views. The more poisoned the graph is, the more helpful neighbors are needed, and the more trust in high-degree nodes.

Table 4: The specific values of k and α which achieve the peak performance on Cora under different perturbation rate

Ptb Rate	0%	5%	10%	15%	20%	35%	50%
k	1	5	7	7	7	7	13
α	-0.5	-0.3	0.3	0.6	0.6	0.7	0.8

5 RELATED WORK

In this section, we present the related literature on robust graph neural networks and graph contrastive learning.

5.1 Robust GNNs

Extensive studies have demonstrated that GNNs are highly fragile to adversarial attacks [7, 40, 51, 54, 55]. The attackers can greatly degrade the performance of GNNs by limitedly modifying the graph data, namely structure and features.

To strengthen the robustness of GNNs, multiple methods have been proposed in the literature, including structure learning [22], adversarial training [41], utilizing Gaussian distributions to represent nodes [49], designing a new message passing scheme driven by l_1-based graph smoothing [28], combining the kNN graph and the original graph [20], and excluding the low-rank singular components of the graph [8]. [13] and [3] study the robustness of GNNs from the breakdown point perspective and propose more robust aggregation approaches.

In the methods mentioned above, one representative type of approach is graph structure learning [22, 30, 45, 52], which aims to detect the potential adversarial edges and assign these edges lower weights or even remove them. ProGNN [22] and GLNN [11] tried to directly optimize the structure by treating it as a learnable parameter, and they introduced some regularizations like sparsity, feature smoothness, and low-rank into the objective. [40] has found that attackers tend to connect two nodes with different features, so they invented Jaccard to prune the edges that link two dissimilar nodes. GNNGuard [45] also modeled the edge weights by raw features. They further calculated the edge pruning probability through a non-linear transformation. GRCN [44] and GAUGM [47] directly computed the weights of the edges by the inner product of representations learned by the classifier, and no additional parameters were needed.

Different from the above-mentioned methods, we aim to leverage task-irrelevant representations, which represent rich properties of nodes and are insensitive to perturbations, to refine the graph structure. Then, we input the well-refined graph into a light robust classifier.

5.2 Graph Contrastive Learning

To learn task-irrelevant representations, we consider using unsupervised methods, which are mainly divided into three categories: GAEs [12, 23], random walk methods [19, 33], and contrastive methods [15, 32, 36, 53]. Both GAEs and random walk methods are suffer from the proximity over-emphasizing [36], and the augmentation scheme in contrastive methods are naturally similar to adversarial attacks.

Graph contrastive methods are proved to be effective in node classification tasks [15, 32, 36, 53]. The first such method, DGI [36], transferred the mutual information maximization [16] to the graph domain. Next, InfoGraph [34] modified DGI's pipeline to make the global representation useful for graph classification tasks. Recently, GRACE [53] and GraphCL [43] adapted the data augmentation methods in vision [4] to graphs and achieved state-of-the-art performance. Specifically, GraphCL generated views by various augmentations, such as node dropping, edge perturbation, attribute masking, and subgraph. However, these augmentations were not designed for adversarial attack scenarios. The edge perturbation method in GraphCL randomly added or deleted edges in the graph, and we empirically prove this random augmentation does not work in Appendix A.5.

Unlike GraphCL and GRACE randomly generated augmentations, STABLE generates robustness-oriented augmentations by a novel recovery scheme, which simulates attacks to make the representations insensitive to the perturbations.

6 CONCLUSION

To overcome the vulnerability of GNNs, we propose a novel defense model, STABLE, which successfully refines the graph structure via more reliable representations. Further, we design an advanced GCN as a downstream classifier to enhance the robustness of GCN. Our experiments demonstrate that STABLE consistently outperforms state-of-the-art baselines and can defend against different types of attacks. For future work, we aim to explore representation learning

in adversarial attack scenarios. The effectiveness of STABLE proves that robust representation might be the key to GNN robustness.

ACKNOWLEDGMENTS

This work is supported by Alibaba Group through Alibaba Innovative Research Program. This work is supported by the National Natural Science Foundation of China under Grant (No.61976204, U1811461, U1836206). Xiang Ao is also supported by the Project of Youth Innovation Promotion Association CAS, Beijing Nova Program Z201100006820062. Yang Liu is also supported by China Scholarship Council. We would like to thank the anonymous reviewers for their valuable comments, and Mengda Huang, Linfeng Dong, Zidi Qin for their insightful discussions.

REFERENCES

[1] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. 2019. Mixhop: Higher-order Graph Convolutional Architectures via Sparsified Neighborhood Mixing. In *ICML*. PMLR, 21–29.

[2] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. 2018. Relational Inductive Biases, Deep Learning, and Graph Networks. *arXiv preprint arXiv:1806.01261* (2018).

[3] Liang Chen, Jintang Li, Qibiao Peng, Yang Liu, Zibin Zheng, and Carl Yang. 2021. Understanding Structural Vulnerability in Graph Convolutional Networks. *arXiv preprint arXiv:2108.06280* (2021).

[4] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A Simple Framework for Contrastive Learning of Visual Representations. In *ICML*.

[5] Yu Chen, Lingfei Wu, and Mohammed Zaki. 2020. Iterative Deep Graph Learning for Graph Neural Networks: Better and Robust Node Embeddings. *NeurIPS* 33.

[6] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. 2021. Adaptive Universal Generalized Pagerank Graph Neural Network. In *ICLR*.

[7] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. 2018. Adversarial Attack on Graph Structured Data. In *ICML*. 1115–1124.

[8] 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 *WSDM*. 169–177.

[9] Bahare Fatemi, Layla El Asri, and Seyed Mehran Kazemi. 2021. SLAPS: Self-Supervision Improves Structure Learning for Graph Neural Networks.

[10] Ben Finkelshtein, Chaim Baskin, Evgenii Zheltonozhskii, and Uri Alon. 2020. Single-Node Attack for Fooling Graph Neural Networks. *arXiv preprint arXiv:2011.03574* (2020).

[11] Xiang Gao, Wei Hu, and Zongming Guo. 2020. Exploring Structure-adaptive Graph Learning for Robust Semi-supervised Classification. In *ICME*. IEEE.

[12] Alberto Garcia Duran and Mathias Niepert. 2017. Learning Graph Representations With Embedding Propagation. In *NeurIPS*, Vol. 30.

[13] Simon Geisler, Daniel Zügner, and Stephan Günnemann. 2020. Reliable Graph Neural Networks via Robust Aggregation. In *NeurIPS*.

[14] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NeruIPS*. 1025–1035.

[15] Kaveh Hassani and Amir Hosein Khasahmadi. 2020. Contrastive Multi-view Representation Learning on Graphs. In *ICML*. PMLR, 4116–4126.

[16] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. 2019. Learning Deep Representations by Mutual Information Estimation and Maximization. In *ICLR*.

[17] Yifan Hou, Jian Zhang, James Cheng, Kaili Ma, Richard TB Ma, Hongzhi Chen, and Ming-Chang Yang. 2019. Measuring and Improving The Use of Graph Information in Graph Neural Networks. In *ICLR*.

[18] Mengda Huang, Yang Liu, Xiang Ao, Kuan Li, Jianfeng Chi, Jinghua Feng, Hao Yang, and Qing He. 2022. AUC-oriented Graph Neural Network for Fraud Detection. In *WWW*.

[19] Glen Jeh and Jennifer Widom. 2003. Scaling Personalized Web Search. In *WWW*.

[20] Wei Jin, Tyler Derr, Yiqi Wang, Yao Ma, Zitao Liu, and Jiliang Tang. 2021. Node Similarity Preserving Graph Convolutional Networks. In *WSDM*. 148–156.

[21] Wei Jin, Yaxin Li, Han Xu, Yiqi Wang, Shuiwang Ji, Charu Aggarwal, and Jiliang Tang. 2020. Adversarial Attacks and Defenses on Graphs: A Review, A Tool and Empirical Studies. In *KDD Explorations*.

[22] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. 2020. Graph Structure Learning for Robust Graph Neural Networks. In *KDD*.

[23] Thomas N Kipf and Max Welling. 2016. Variational Graph Auto-Encoders. *NIPS Workshop on Bayesian Deep Learning* (2016).

[24] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

[25] Ruoyu Li, Sheng Wang, Feiyun Zhu, and Junzhou Huang. 2018. Adaptive Graph Convolutional Neural Networks. In *AAAI*, Vol. 32.

[26] Yaxin Li, Wei Jin, Han Xu, and Jiliang Tang. 2020. Deeprobust: A Pytorch Library for Adversarial Attacks and Defenses. *arXiv preprint arXiv:2005.06149* (2020).

[27] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2016. Gated Graph Sequence Neural Networks. *ICLR*.

[28] Xiaorui Liu, Wei Jin, Yao Ma, Yaxin Li, Hua Liu, Yiqi Wang, Ming Yan, and Jiliang Tang. 2021. Elastic Graph Neural Networks. In *ICML*. PMLR, 6837–6849.

[29] 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 *WWW*. 3168–3177.

[30] Dongsheng Luo, Wei Cheng, Wenchao Yu, Bo Zong, Jingchao Ni, Haifeng Chen, and Xiang Zhang. 2021. Learning to Drop: Robust Graph Neural Network via Topological Denoising. In *WSDM*. 779–787.

[31] Miller McPherson, Lynn Smith-Lovin, and James M Cook. 2001. Birds of a Feather: Homophily in Social Networks. *Annual review of sociology* 27, 1 (2001), 415–444.

[32] Zhen Peng, Wenbing Huang, Minnan Luo, Qinghua Zheng, Yu Rong, Tingyang Xu, and Junzhou Huang. 2020. Graph Representation Learning via Graphical Mutual Information Maximization. In *WWW*.

[33] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online Learning of Social Representations. In *KDD*. 701–710.

[34] Fan-Yun Sun, Jordan Hoffmann, Vikas Verma, and Jian Tang. 2020. Infograph: Unsupervised and Semi-supervised Graph-level Representation Learning via Mutual Information maximization. In *ICLR*.

[35] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph Attention Networks. In *ICLR*.

[36] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2018. Deep Graph Infomax. *arXiv preprint arXiv:1809.10341* (2018).

[37] Xiao Wang, Meiqi Zhu, Deyu Bo, Peng Cui, Chuan Shi, and Jian Pei. 2020. Am-gcn: Adaptive Multi-channel Graph Convolutional Networks. In *KDD*. 1243–1253.

[38] Marcin Waniek, Tomasz P Michalak, Michael J Wooldridge, and Talal Rahwan. 2018. Hiding Individuals and Communities in A Social Network. *Nature Human Behaviour* 2, 2 (2018), 139–147.

[39] Olivia Wiles, Sven Gowal, Florian Stimberg, Sylvestre Alvise-Rebuffi, Ira Ktena, Taylan Cemgil, et al. 2022. A Fine-grained Analysis on Distribution Shift. In *ICLR*.

[40] 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. *IJCAI*.

[41] Kaidi Xu, Hongge Chen, Sijia Liu, Pin-Yu Chen, Tsui-Wei Weng, Mingyi Hong, and Xue Lin. 2019. Topology Attack and Defense for Graph Neural Networks: An Optimization Perspective. *IJCAI*.

[42] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful Are Graph Neural Networks?. In *ICLR*.

[43] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. 2020. Graph Contrastive Learning With Augmentations. *NeurIPS*.

[44] Donghan Yu, Ruohong Zhang, Zhengbao Jiang, Yuexin Wu, and Yiming Yang. 2020. Graph-revised Convolutional Network. In *ECML*. Springer, 378–393.

[45] Xiang Zhang and Marinka Zitnik. 2020. GNNGuard: Defending Graph Neural Networks Against Adversarial Attacks. *NeurIPS*.

[46] Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2020. Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering* (2020).

[47] Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, and Neil Shah. 2021. Data Augmentation for Graph Neural Networks. In *AAAI*.

[48] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2020. Graph Neural Networks: A Review of Methods and Applications. *AI Open* 1 (2020), 57–81.

[49] Dingyuan Zhu, Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2019. Robust Graph Convolutional Networks Against Adversarial Attacks. In *KDD*. 1399–1407.

[50] Xiaoqian Zhu, Xiang Ao, Zidi Qin, Yanpeng Chang, Yang Liu, Qing He, and Jianping Li. 2021. Intelligent financial fraud detection practices in post-pandemic era. *The Innovation* 2, 4 (2021), 100176.

[51] Yulin Zhu, Yuni Lai, Kaifa Zhao, Xiapu Luo, Mingquan Yuan, Jian Ren, and Kai Zhou. 2022. BinarizedAttack: Structural Poisoning Attacks to Graph-based Anomaly Detection. In *ICDE*.

[52] Yanqiao Zhu, Weizhi Xu, Jinghao Zhang, Qiang Liu, Shu Wu, and Liang Wang. 2021. Deep Graph Structure Learning for Robust Representations: A Survey. *arXiv preprint arXiv:2103.03036*.

[53] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. 2020. Deep Graph Contrastive Representation Learning. In *ICML Workshop on Graph Representation Learning and Beyond*. http://arxiv.org/abs/2006.04131

[54] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. 2018. Adversarial Attacks on Neural Networks for Graph Data. In *KDD*. 2847–2856.

[55] Daniel Zügner and Stephan Günnemann. 2019. Adversarial Attacks on Graph Neural Networks via Meta Learning. In *ICLR*.

A APPENDIX

A.1 Algorithm

The overall training algorithm is shown in Algorithm 1. In line 2, we roughly pre-process $\mathcal{G}$ by removing some potential perturbations. From lines 3 to 10, we utilize a contrastive learning model with robustness-oriented augmentations to obtain the node representations. In lines 11 and 12, we refine the graph structure based on the representations learned before. from line 13 to 20, we train the classifier GCN* on G^*.

Algorithm 1: STABLE

Input: Graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{X}\}$, Labels $\mathcal{Y}_L$, pre-process threshold t_1, recovery probability p, pruning threshold t_2, number of adding helpful neighbors k, advanced GCN parameters α and β, training epochs N_{epoch}

Output: The predicted labels of $\mathcal{V}_U$

1 Initialize parameters ϕ, ω and θ;
2 Roughly pre-process $\mathcal{G}$ to obtain $\mathcal{G}^P$ using Eq. (4) and t_1;
3 Generate augmentations $\mathcal{G}_1^P$ to $\mathcal{G}_M^P$ using Eq. (5);
4 Generate $\tilde{\mathcal{G}}^P$ by shuffling the node features of $\mathcal{G}^P$;
5 **while** *not converge* **do**
6 Compute $\mathbf{H}$, $\tilde{\mathbf{H}}$, $\mathbf{H}_1$, $\mathbf{H}_2$..., and $\mathbf{H}_M$ using Eq. (6);
7 Compute the global representations of $\mathcal{G}_1^P$ to $\mathcal{G}_M^P$ using Eq. (7);
8 Compute the contrastive learning objective using Eq.(8);
9 Update ϕ and ω by $\frac{\partial \mathcal{L}_C}{\partial \phi}$ and $\frac{\partial \mathcal{L}_C}{\partial \omega}$;
10 **end**
11 Prune the strcutrue using Eq. (9) and Eq.(10);
12 Add helpful edges using Eq. (11);
13 **for** *e=1, ..., N_{epoch}* **do**
14 **for** *t=1, 2* **do**
15 **for** $v_i \in \mathcal{V}$ **do**
16 Compute h_i using Eq. (12);
17 **end**
18 **end**
19 Compute $\mathcal{L}_G$ using Eq. (13);
20 Update θ by $\frac{\partial \mathcal{L}_G}{\partial \theta}$
21 **end**
22 return argmax$(f_\theta(\mathbf{H}, \mathsf{A}^*)_{\mathcal{V}_U}$, dim=1);

A.2 Datasets

Following [22, 54], we only consider the largest connected connected component (LCC). The statistics is listed in Tabel 5. There is no features available in Polblogs. Following [22, 28] we set the feature matrix to be a $n \times n$ identity matrix. The results of GNNGuard and Jaccard on Polblogs are not available because the cosine similarity of the identity matrix is meaningless. For PubMed dataset, we use the attacked graphs provided provided by [21].

Table 5: Dataset statistics. We only consider the largest connected component (LCC).

Datasets	N_{LCC}	E_{LCC}	Classes	Features
Cora	2,485	5,069	7	1433
Citeseer	2,110	3,668	6	3703
Polblogs	1,222	16,714	2	/
PubMed	19717	44338	3	500

A.3 Baselines

- **GCN** [24]: GCN is a popular graph convolutional network based on spectral theory.
- **RGCN** [49]: RGCN utilizes gaussian distributions to represent node and uses a variance-based attention mechanism to remedy the propagation of adversarial attacks.
- **Jaccard** [40]: Since attacks tend to link nodes with different labels, Jaccard prune edges which connect two dissimilar nodes.
- **GNNGuard** [45]: GNNGuard utilizes cosine similarity to model the edge weights and then calculates edge pruning probability through a non-linear transformation.
- **GRCN** [44]: GRCN models edge weights by taking inner product of embeddings of two end nodes.
- **ProGNN** [22]: ProGNN treats the adjacency matrix as learnable parameters and directly optimizes it with three regularizations, *i.e.*, feature smoothness, low-rank and sparsity.
- **SimpGCN** [20]: SimpGCN utilizes a kNN graph to keep the nodes with similar features close in the representation space and a self-learning regularization to keep the nodes with dissimilar features remote.
- **Elastic** [28]: Elastic introduces ℓ_1-norm to graph signal estimator and proposes elastic message passing which is derived from one step optimization of such estimator. The local smoothness adaptivity enables the Elastic GNNs robust to structural attacks.
- **MetaAttack** [55]: MetaAttack uses meta-gradients to solve the bilevel problem underlying poisoning attacks, essentially treating the graph as a hyperparameter to optimize.
- **DICE** [38]: Disconnect Internally, connect externally.
- **Random**: Inject random structure noise.

A.4 Implementation Details

We use DeepRobust, an adversarial attack repository [26], to implement all the attack methods, RGCN, ProGNN, SimpGCN, and Jaccard. GNNGuard, Elastic, and GCN are implemented with the code provided by the authors.

For each graph, we randomly split the nodes into 10% for training, 10% for validation, and 80% for testing. Then we generate attacks on each graph according to the perturbation rate, and all the hyper-parameters in attack methods are the same with the authors' implementation. For all the defense models, we report the average accuracy and standard deviation of 10 runs.

All the hyper-parameters are tuned based on the loss and accuracy of the validation set. For RGCN, the hidden dimensions are tuned from {16, 32, 64, 128}. For Jaccard, the Jaccard Similarity

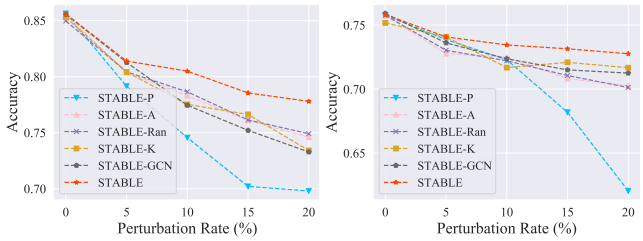
threshold are tuned from {0.01, 0.02, 0.03, 0.04, 0.05}. For GNNGuard, we follow the author's settings, which contains $P_0 = 0.5$, $K = 2$, $D_2 = 16$ and $dropout = 0.5$. For ProGNN and SimpGCN, following [20], we use the default hyper-parameter settings in the authors' implementation. For Elastic, the propagation step K is tuned from {3, 5, 10}, and the parameter λ_1 and λ_2 are tuned from {0, 3, 6, 9}.

For our work, t_1 is Jaccard Similarity threshold in this work and tuned from{0.0, 0.01, 0.02, 0.03, 0.04, 0.05}, the recovery portion p is fixed at 0.2, t_2 is tuned from {0.1, 0.2, 0.3}, k is tuned from {1, 3, 5, 7, 11, 13}, α is tuned from -0.5 to 3, and β is fixed at 2. We fix $M = 2$ because we find that two augmentation views are good enough. The other parameters in GCN^* follows the setting in [24].

A.5 Ablation Study (RQ4)

We conduct an ablation study to examine the contributions of different components in STABLE:

- **STABLE-P**: STABLE without rough pre-process.
- **STABLE-A**: STABLE without augmentations.
- **STABLE-Ran**: STABLE with random augmentations. We generate the views by randomly removing or adding edges.
- **STABLE-K**: We only prune edges in the refining phase.
- **STABLE-GCN**: Replace advanced GCN with GCN.



(a) Cora (b) Citeseer

Figure 5: Comparisons between STABLE and its variants.

The accuracy on Cora and Citeseer under MetaAttack is illustrated in Fig. 5. It is clearly observed that the STABLE-P perform worst, and it is in line with our point that $\mathcal{G}^P$ is much cleaner than $\mathcal{G}$. STABLE-Ran and STABLE-A perform very closely, and the gap between them and STABLE widens as the perturbation rate increases. We can conclude that the robustness-oriented augmentation indeed works. It is worth noting that STABLE outperforms STABLE-K more significantly as the perturbation rate rising, as our expectation. We argue that adding helpful neighbors can diminish the impact of harmful edges, especially on heavily poisoned graphs.

To further verify the robustness of advanced GCN, we present the performance of SimpGCN* and Jaccard* in Table 6. Jaccard and SimpGCN both contain the vanilla GCN, so we can easily define two variants by replacing the GCN with the advanced GCN. As shown in Table 6, the variants of Jaccard and SimpGCN show more robust than the original models. The improvement of SimpGCN is more significant might because of the pruning strategy in Jaccard, which already removes some adversarial edges.

Table 6: Classification accuracy(%) on Cora under different perturbation rates. The asterisk indicates that the GCN part of this model is replaced with advanced GCN.

Datasets	Ptb rate	Jaccard	Jaccard*	SimpGCN	SimpGCN*
Cora	0%	**81.79**	81.11	**83.77**	83.64
	5%	80.23	**80.57**	78.98	**80.45**
	10%	74.65	**76.99**	75.07	**78.04**
	15%	74.29	**76.32**	71.42	**75.31**
	20%	73.11	**73.42**	68.90	**73.29**
	35%	66.11	**68.79**	64.87	**71.15**
	50%	58.08	**64.06**	51.94	**65.63**

A.6 Sensitivity

We explore the sensitivity of t_1 and t_2 for STABLE. The performance change of STABLE is illustrate in Figure 6. In fact, for Cora, we fixed t_1 at 0.03 and t_2 at 0.2 to achieve the best performance under different perturbation rate. For t_1, it is worth noting that, the performance on pre-processed graph is consistently higher than on the original graph($t_1 = 0$). $t_2 = 0$ implies that no edge is pruned in the refining step, and the performance is still competitive might due to the good representations and helpful edge insertions.

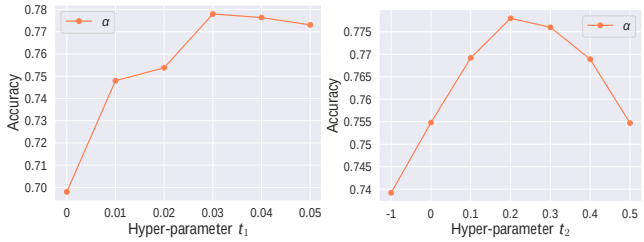


Figure 6: Parameter sensitivity analysis on Cora.