Adversarially Robust Neural Architecture Search for Graph Neural Networks

Beini Xie1*  Heng Chang1*  Ziwei Zhang1  Xin Wang1†  Daixin Wang2  Zhiqiang Zhang2  Rex Ying3  Wenwu Zhu1†  Ant Group  Yale University

1†Tsinghua University  2Ant Group  3Yale University

Abstract

Graph Neural Networks (GNNs) obtain tremendous success in modeling relational data. Still, they are prone to adversarial attacks, which are massive threats to applying GNNs to risk-sensitive domains. Existing defensive methods neither guarantee performance facing new data/tasks or adversarial attacks nor provide insights to understand GNN robustness from an architectural perspective. Neural Architecture Search (NAS) has the potential to solve this problem by automating GNN architecture designs. Nevertheless, current graph NAS approaches lack robust design and are vulnerable to adversarial attacks. To tackle these challenges, we propose a novel Robust Neural Architecture search framework for GNNs (G-RNA). Specifically, we design a robust search space for the message-passing mechanism by adding graph structure mask operations into the search space, which comprises various defensive operation candidates and allows us to search for defensive GNNs. Furthermore, we define a robustness metric to guide the search procedure, which helps to filter robust architectures. In this way, G-RNA helps understand GNN robustness from an architectural perspective and effectively searches for optimal adversarial robust GNNs. Extensive experimental results on benchmark datasets show that G-RNA significantly outperforms manually designed robust GNNs and vanilla graph NAS baselines by 12.1% to 23.4% under adversarial attacks.

1. Introduction

Graph Neural Networks are well-known for modeling relational data and are applied to various downstream real-world applications like recommender systems [37], knowledge graph completion [25], traffic forecasting [6], drug production [45], etc. Meanwhile, like many other deep neural networks, GNNs are notorious for their vulnerability under adversarial attacks [33], especially in risk-sensitive domains, such as finance and healthcare. Since GNNs model node representations by aggregating the neighborhood information, an attacker could perform attacks by perturbing node features and manipulating relations among nodes [55]. For example, in the user-user interaction graph, a fraudster may deliberately interact with other important/fake users to mislead the recommender system or fool credit scoring models [36].

A series of defense methods on graph data have been developed to reduce the harm of adversarial attacks. Preprocessing-based approaches like GCN-SVD [8] and GCN-Jaccard [38] conduct structure cleaning before training GNNs, while attention-based models like RGCN [51] and GNN-Guard [46] learn to focus less on potential perturbed edges. However, these methods rely on prior knowledge of the attacker. For example, GCN-SVD leverages the high-rank tendency of graph structure after Nettack [53], and GCN-Jaccard depends on the homophily assumption on the graph structure. As a result, current approaches may fail to adapt to scenarios when encountering new data and tasks or when new attack methods are proposed. Additionally, previous methods largely neglect the role of GNN architectures in defending against adversarial attacks, lacking an architectural perspective in understanding GNN robustness.

In order to reduce human efforts in neural architecture designs, Neural Architecture Search (NAS) has become increasingly popular in both the research field and industry. Though NAS has the potential of automating robust GNN designs, existing graph NAS methods [1, 10, 24, 49, 50] are inevitably susceptible to adversarial attacks since they do not consider adversarial settings and lack robustness designs [48]. Therefore, how to adopt graph NAS to search for optimal robust GNNs in various environments, and in the meantime, fill the gap of understanding GNN robustness from an architectural perspective, remains a huge challenge.

To address the aforementioned problems and to understand GNN robustness from an architectural perspective, we propose a novel Robust Neural Architecture search framework for Graph neural networks (G-RNA), which is the first attempt to exploit powerful graph neural architecture search in robust GNNs, to our best knowledge. Specifically, we first design a novel, expressive, and robust search space with

*Equal contribution.
†Corresponding authors.
The overall framework of G-RNA. A clean graph, the supernet built upon our robust search space is trained in a single-path one-shot way. Then, the attack proxy produces several adversarial samples based on the clean graph and we search for robust GNNs with the proposed robustness metric. Finally, we evaluate the optimal robust GNN on graphs perturbed by the attacker.

Figure 1. The overall framework of G-RNA. Given a clean graph, the supernet built upon our robust search space is trained in a single-path one-shot way. Then, the attack proxy produces several adversarial samples based on the clean graph and we search for robust GNNs with the proposed robustness metric. Finally, we evaluate the optimal robust GNN on graphs perturbed by the attacker.

We develop a robust neural architecture search framework for GNNs, which considers robust designs in graph neural architecture search for the first time to the best of our knowledge. Based on this framework, we can understand adversarial robustness for GNNs from an architectural perspective.

We propose a novel robust search space by designing defensive operation candidates to automatically select the most appropriate defensive strategies when confronting perturbed graphs. Besides, we design a robustness metric and adopt an evolutionary algorithm together with a single-path one-shot graph NAS framework to search for the optimal robust architectures.

Extensive experimental results demonstrate the efficacy of our proposed method. G-RNA outperforms state-of-the-art robust GNNs by 12.1% to 23.4% on benchmark datasets under heavy poisoning attacks.

2. Related Work

2.1. Adversarial Robustness on Graph Data

Despite the wide success of GNNs in various applications [43, 44, 47], GNNs are shown vulnerable to adversarial attacks [2–4, 33, 39, 53], i.e., slight perturbations to graph can lead to sharp performance decrease. Following the literature, adversarial attacks have various taxonomies according to the attacker’s knowledge (white-box attack and black-box attack), perturbation type (structure attack and feature attack), attack stage (evasion attack and poisoning attack), and targeted nodes (targeted attack and non-targeted attack).

In response to adversarial attacks, several defensive models have been proposed to enhance the robustness of GNNs. Graph pre-processing methods identify and rectify potential structural perturbations before the GNN model training. For example, with the assumption of feature smoothness, GCN-Jaccard [38] removes edges that have a low Jaccard similarity. Observing the high-rank tendency of the adjacency matrix under Nettack, GCN-SVD [8] reconstructs the adjacency matrix via its low-rank approximation. Graph attention methods aim to learn fewer attention weights on susceptible edges/features. For example, RGCN [51] uses Gaussian distribution for hidden layer node representations and calculates attention based on their variance. Pro-GNN [20] jointly learns graph structure and model parameters by keeping a low-rank and sparse adjacency matrix as well as feature smoothness. GNN-Guard [46] learns to focus more on edges between similar nodes and pruning edges between unrelated nodes. VPN [19] defenses by re-weighting edges from graph...
powering considering $r$-hop neighbors. Besides, graph certificate robustness methods [18, 35] provide a theoretical guarantee for graphs to be certified as robust under perturbation budgets. However, this branch is out of the scope of this work and we leave the comparison as a future work.

All previous studies rely heavily on manual designs and thus cannot adapt to new data, tasks, or adversarial attacks. Besides, existing methods neglect the inherent robustness of GNN architectures. On the contrary, our proposed method with specifically designed search space on graph structure can automatically search for the optimal robust GNN for different data and tasks.

### 2.2. Graph Neural Architecture Search

Neural Architecture Search (NAS) is a proliferate research direction that automatically searches for high-performance neural architectures and reduces the human efforts of manually-designed architectures. NAS on graph data is challenging because of the non-Euclidean graph property and special neural architectures [42, 48]. GraphNAS [10] uses the recurrent network as the controller to generate GNN architectures and adopts reinforcement learning to search for optimal architectures. In order to conduct an efficient search, differentiable NAS approaches [14, 26] jointly optimize the model weights and architecture parameters. DSS [24] proposes a differentiable one-shot graph NAS and dynamically updates the search space. SANE [49] also utilizes a differentiable search algorithm and builds GNN architectures with the Jumping Knowledge Network (JK-Net) backbone [41]. GNAS [1] reconstructs GNNs with the designed GNN Paradigm and learns the optimal message-passing depth as well. GASSO [30] uses graph structure learning as a denoising process in the differentiable architecture searching process. Graph NAS is also used in complex graph data such as heterogeneous graphs [7] and temporal graphs [29]. However, the existing graph NAS methods do not consider robustness against adversarial attacks.

### 2.3. Robust Neural Architecture Search

Robust neural architecture search exploits NAS to search for adversarially robust neural architectures. Since there is no related work for robust NAS on graph data, we review two remotely related papers on computer vision. RobNets [13] is the first work to explore architecture robustness. Through one-shot NAS, RobNets finetune architecture candidates via adversarial training and then sample more robust architectures. DSRNA [16] proposes two differentiable metrics which help to search robust architectures by a differentiable search algorithm. Our work is neither based on adversarial training nor adopts continuous relaxation for architecture parameters. To conclude, our work differs in that we consider a disparate search space tailored for graph data and leverage a different search algorithm.

### 3. Preliminaries

#### 3.1. Graph Neural Networks

Let $G = (A, X)$ denote a graph with $N$ nodes, where $A \in \mathbb{R}^{N \times N}$ is the adjacency matrix and $X \in \mathbb{R}^{N \times D_0}$ is the corresponding feature matrix. For node $i$, its neighborhood is denoted as $N(i)$.

Graph Neural Networks take the graph data as input and output node/graph representations to perform downstream tasks like node classification and graph classification. Typically, for node classification tasks with $C$ labels, we calculate:

$$z_i = (f_\alpha(A, X))_i,$$

where $z_i \in \mathbb{R}^C$ is the prediction vector for node $i$, $f_\alpha$ denotes the graph neural network based on architecture $\alpha$. The design of GNNs could be divided into intra-layer design, inter-layer design, and learning configurations [42]. Intra-layer design often follows the message-passing paradigm [11]: nodes representation is updated by aggregating neighborhood information. A general formula for updating node representation in GNNs is denoted as

$$h_i^{(l)} = \sigma \left( W^{(l)} \text{Comb} \left( h_i^{(l-1)}, \text{Aggr} (c_{ij}^{(l)} h_j^{(l-1)}, j \in N(i)) \right) \right),$$

where $h_i^{(l)}$ denotes the node representation for node $i$ in the $l$-th hidden layer, $c_{ij}^{(l)}$ is the correlation coefficient between node $i$ and $j$, $N(i) = \{i\} \cup N(i)$ represents the neighborhood of node $i$ with the self-loop, Aggr$(\cdot)$ is the function to aggregate neighborhood information, Comb$(\cdot)$ aims for combining self- and neighbor-information, and $\sigma(\cdot)$ is the activation function.

Besides the GNN layer design, how to connect different hidden layers is also critical. Some GNNs directly use the last hidden layer output as the prediction, while pre-processing layers, post-processing layers, and skip connections could also be added [22]. As for learning configurations, they are hyper-parameters for training GNNs like the learning rate, the optimizer, etc.

#### 3.2. Graph Neural Architecture Search

In general, graph NAS could be formulated as the following bi-level optimization problem:

$$\alpha^* = \arg \max_{\alpha \in \mathcal{O}} \text{Acc}_{val}(W^*(\alpha), \alpha)$$

s. t. $W^*(\alpha) = \arg \min_{W} \mathcal{L}_{train}(W, \alpha),$

where Eq. (3) is the upper-level optimization problem to find the best architecture $\alpha^*$ in the search space $\mathcal{O}$, and Eq. (4) is the lower-level problem to calculate optimal model weights $W$ for one particular architecture $\alpha$. Acc$\text{val}$ represents the
prediction accuracy on the validation set and $L_{train}$ is the classification cross-entropy loss on the training set.

Existing graph NAS methods [1, 10, 12, 48, 50] design their search space following the message-passing scheme in Eq. (1). The most commonly used correlation coefficient operations are provided in Appendix B.2. NAS methods can search all components in Eq. (1) such as the aggregation function, correlation coefficients, and activation functions as well as hyper-parameters like hidden size and learning configurations. In this paper, we mainly consider searching for architectural designs.

4. Robust Graph Neural Architecture Search

In this section, we first formulate the problem of graph robust NAS. Then, we introduce our novel and expressive search space with defensive operation candidates, namely graph structure masks. Based on the designed search space, we build a supernet containing all possible architectures and train it in a single-path one-shot way. Finally, we introduce our proposed robustness metric and describe the search process exploiting the evolutionary algorithm.

4.1. Problem formulation

Given a search space $O$, we aim to find the optimal architecture $\alpha^* \in O$ with both high prediction accuracy and high adversarial robustness. We formulate the problem of robust neural architecture search for GNNs as:

$$\alpha^* = \arg \max_{\alpha \in O} ACC_{val}(\alpha) + \lambda R(\alpha),$$

where $R(\cdot)$ is the robustness metric, and $\lambda$ is a hyper-parameter balancing the model accuracy and robustness.

4.2. Search Space for Robust GNNs

We design a fine-grained search space following the message-passing paradigm. In total, there are six adjustable components in our GNN architecture: the graph structure mask, the nodes correlation coefficient, the neighbor aggregator, the combine function, the skip connection, and the layer aggregator. The first four components belong to the intra-layer operations, while the rest two components are inter-layer operations.

**Intra-layer Operations.** Inside the $l$-th message-passing layer, the defensive operation $D \in O_D$ is firstly adopted to construct a graph structure mask and reconstruct the graph structure:

$$M^{(l)} = D(A^{(l-1)}), A^{(l)} = A \odot M^{(l)}$$

where $M^{(l)} = \{m_{ij}\}$ is the graph structure mask matrix and $A^{(l)}$ denotes the graph structure in the $l$-th layer. $\odot$ is the Hadamard product and $A^{(0)} = A$. Each element $m_{ij} \in [0, 1]$ is the mask score between node $i$ and node $j$ where $m_{ij} = 0$ indicates a complete edge pruning and $m_{ij} = 1$ means no modification to the original edge. The defensive operation aims to assign fewer weights to potential perturbed edges.

Inspired by the success of current defensive approaches [33], we conclude the properties of operations on graph structure for robustness and design representative defensive operations in our search space accordingly. In this way, we can choose the most appropriate defensive strategies when confronting perturbed graphs. To our best knowledge, this is the first time the search space to be designed with a specific purpose to enhance the robustness of GNNs. Specifically, we include five graph structure mask operations in the search space. Identity keeps the same graph structure as the previous layer. Low Rank Approximation (LRA) reconstructs the adjacency matrix in the $l$-th layer from the top-$r$ components of singular value decomposition from adjacency matrix in the previous layer. Node Feature Similarity (NFS) deletes edges that have small Jaccard similarities among node features. Neighbor Importance Estimation (NIE) updates mask values with a pruning strategy based on quantifying the relevance among nodes. Variable Power Operator (VPO) forms a variable power graph from the original adjacency matrix weighted by the parameters of influence strengths.

Moreover, more operations like graph structure learning [52] could be integrated into the graph structure mask operations. The formula for mask operations is shown in Table 1. We limit LRA and NFS in the first layer (only for pre-processing) and exploit the other three operations for all layers. It is worth mentioning that the graph structure mask candidates could be easily extended to other methods that deal with the graph structure, like denoising methods. Mask operations will not occlude the later message-passing process. They could be seen as orthogonal operations to other operations like the correlation coefficient.

For the other three components in the intra-layer, our choices for correlation coefficients follow the literature summarized in Table 4 in Appendix. Based on the masked graph

<table>
<thead>
<tr>
<th>Operation</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity</td>
<td>$M^{(l)} = A^{(l-1)}$</td>
</tr>
<tr>
<td>LRA</td>
<td>$A^{(l-1)} = U^{(l-1)} S^{(l-1)} (V^{(l-1)})^T$, $M^{(l)} = U^{(l-1)} S^{(l-1)} (V^{(l-1)})^T$</td>
</tr>
<tr>
<td>NFS</td>
<td>$m_{ij}^{(l)} = \begin{cases} 0, &amp; \text{if } a_{ij}^{(l-1)} &gt; 0 \text{ and } J_{ij} &lt; \tau \ a_{ij}^{(l-1)}, &amp; \text{otherwise} \end{cases}$</td>
</tr>
<tr>
<td>NIE</td>
<td>$m_{ij}^{(l)} = \beta m_{ij}^{(l-1)} + (1 - \beta) \hat{a}_{ij}^{(l)}$</td>
</tr>
<tr>
<td>VPO</td>
<td>$M^{(l)} = \sum_{v=1}^{V} \theta_v (A^{(l-1)})^v$</td>
</tr>
</tbody>
</table>
structure $A^{(l)}$ and correlation coefficients $\{e_{ij}^{(l)}\}$, a neighbor aggregator $\text{Aggr} \in O_{\text{aggr}}$ is used to aggregate neighborhood representations. Afterwards, a combine function $\text{Comb} \in O_{\text{comb}}$ merges self- and neighbor-message. Here, we explore two typical approaches to combine self-representation and neighbor-message, namely GraphSAGE [15] and GIN [40]. GraphSAGE conducts different feature transformations for node representation and neighborhood information. GIN first performs weighted sum for self- and neighbor-message, then uses multi-layer perceptron to improve GNN’s expressive power. After combining messages, the activation function $\sigma(\cdot)$ is applied. Overall, the hidden representation for node $i$ in the $l$-th layer is calculated as

$$h_i^{(l)} = \sigma \left( W^{(l)} \text{Comb} \left( h_i^{(l-1)}, \text{Aggr} \left( e_{ij}^{(l)} h_j^{(l-1)}, j \in \tilde{N}(i) \right) \right) \right).$$  

(7)

The node representation is initialized as node features, i.e., $h^{(0)} = X$.

**Inter-layer Operations.** Following the idea of JK-Net [41] and SANE [49], we aggregate node representations in intermediate layers via the layer aggregator $\text{Layer}_{\text{Aggr}} \in O_{\text{layer}}$. The skip operation $\text{Skip} \in O_{\text{skip}}$ decides the skip connection to the final layer aggregator.

$$z_i = \text{Layer}_{\text{Aggr}} \left( \text{Skip}(h_i^{(1)}), ..., \text{Skip}(h_i^{(L)}) \right),$$

(8)

where $L$ is the maximum number of message-passing layers. With the JK-Net backbone, we choose the optimal model depth by setting a maximal depth. For example, if we set $L = 4$ and the skip operations are $\{\text{Identity}, \text{Zero}, \text{Identity}, \text{Zero}\}$, then the selected optimal number of message-passing layers is 3. After obtaining the final node representations, we add several fully connected layers to conduct classification.

We summarize all components and their candidate operations in Table 2. Our search space is expressive and, more importantly, has improved defense capability. As shown in Table 5 in Appendix, our search space could recover some classic manually designed GNNs and state-of-the-art robust GNNs like GCN-SVD, GCN-Jaccard, GNN-Guard, and VPN.

**Supernet Training.** With the proposed search space, we construct a supernet and train it in a single-path one-shot way [14]. The supernet we built contains all possible architectures based on our search space, also called the one-shot model. In the supernet, each architecture behaves as a single path. Unlike graph NAS methods such as GNAS [1] or DSS [24] that use continuous relaxation for architecture parameters, we train the supernet via uniform single path sampling [14]. During the training process, we uniformly sample one path to update weights each time. Afterward, we search for optimal architectures through the evolutionary algorithm without further training steps.

### 4.3. Measuring Robustness

This part will introduce how to measure the architecture’s robustness and the specific search process using the evolutionary algorithm with our robustness metric.

Intuitively, the performance of a robust GNN should not deteriorate too much when confronting various perturbed graph data. Let $T_{\Delta}$ denote the attacker of the graph data with perturbation budget $\Delta$. We define the robustness metric as

$$R(A, f) = -\mathbb{E}_{A'} \left[ \frac{1}{N} \sum_{i=1}^{N} D_{KL}(f(A)_i || f(A'_i)) \right],$$

(9)

where $f$ indicates the GNN, $f(A)_i$ is the probability prediction vector for node $i$ and $D_{KL}(p||q) = \sum_i p_i \log \frac{p_i}{q_i}$ denotes the Kullback-Leibler (KL) divergence between two distribution $p$ and $q$. Here, we use KL distance to measure the prediction difference given clean and perturbed data. The choice of our attack proxy $T$ varies from simple random attack [5] or DICE [54] to advanced attack methods like Mettack [54] or Nettack [53]. Within the same perturbation budget $\Delta$, our attack proxy generates $T$ perturbed graph structure $\{A'_i\}_{i=1}^T$. Eq. (9) could be approximately computed as

$$R(A, f) \approx -\frac{1}{TN} \sum_{t=1}^{T} \sum_{i=1}^{N} (D_{KL}(f(A)_i || f(A'_i)_i)).$$

(10)

A larger $R(A, f)$ indicates a smaller distance between the prediction of clean data and the perturbed data, and consequently, more robust GNN architectures.

---

Table 2. The six components in search space and corresponding candidate operations.

<table>
<thead>
<tr>
<th>Component</th>
<th>Candidate Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O_D$</td>
<td>Identity, LRA, NFS, VPO, NIE</td>
</tr>
<tr>
<td>$O_e$</td>
<td>Identity, GCN, GAT, GAT-Sym, Cos, Linear, Gene-Linear</td>
</tr>
<tr>
<td>$O_{aggr}$</td>
<td>Sum, Mean, Max</td>
</tr>
<tr>
<td>$O_{comb}$</td>
<td>Identity, GIN, SAGE</td>
</tr>
<tr>
<td>$O_{skip}$</td>
<td>Identity, Zero</td>
</tr>
<tr>
<td>$O_{layer}$</td>
<td>Concat, Max, LSTM</td>
</tr>
</tbody>
</table>
In order to illustrate the effectiveness of the proposed robustness metric, we randomly choose 300 architectures and calculate their robustness metric value and evaluate their performance decrease after 5% structural perturbations in the Cora dataset (the experimental details are described in Sec. 5.2). Fig. 2a displays the distribution of $R$ in log scale while Fig. 2b shows the mean accuracy decrease for different $R$ intervals. The accuracy decrease could be regarded as a ground-truth measurement of the actual robustness. From Fig. 2b, we could see that the mean accuracy decrease has a negative correlation with respect to the robustness metric. When the robustness metric is relatively small, the accuracy may be reduced by as large as 10%. However, when the robustness metric is large, the accuracy decrease shrinks to about 3.5%. This phenomenon indicates that our robustness metric could successfully filter robust architectures.

**Evolutionary Search Algorithm.** In our work, we adopt an evolutionary algorithm to search for optimal robust graph architectures with the proposed robustness metric. Instead of training each candidate architecture from scratch, we leverage the evolutionary algorithm only for inference and search. The weights for all architectures are fixed as those learned in the supernet training phase. In one search epoch, we select top-k robust candidates via the fitness function $ACC_{val}(\alpha) + \lambda R(\alpha)$. Crossover and mutation are followed to generate children architectures from population candidates. We show the search algorithm in Appendix B.4.

5. Experiments

In this section, we conduct experiments to verify our proposed method by evaluating the selected architecture on perturbed graphs. Also, we visualize the performance for diverse operations to better understand GNN architecture’s robustness. Additional experimental results including defensive performance under targeted attack, evaluation on heterophily graphs, and sensitivity analysis for the hyper-parameter $\lambda$ in our proposed robustness metric could be found in Appendix C. The details of the experimental setting are deferred to the Appendix D.

### 5.1. Semi-supervised Node Classification Task

In the semi-supervised node classification task, we perform non-targeted poisoning structural attacks adopting Metattack [34] and evaluate GNN robustness based on perturbed graphs. We vary the proportion of changed edges from 0% to 25% and calculate the retrained accuracy on perturbed graphs. For each setting, we experiment 10 times and report the average results and the standard deviation. The final defense results are summarized in Table 3. More information on searched architectures by G-RNA under attacks is provided in Appendix C.3.

Overall, our G-RNA successfully outperforms all baselines under adversarial attacks. We could see an apparent increase in perturbation performance when confronting heavily poisoning attacks (e.g. when the proportion of changed edges is 15% or more). For instance, the defense performance of G-RNA under 25% structural perturbations exceeds that of vanilla GCN by 23.4%, 20.9%, and 12.1%, respectively. For NAS-based methods, optimal architectures searched on clean data are evaluated on both clean and perturbed graphs to test their robustness and conduct a fair comparison. We could see that there exists a trade-off between architecture accuracy and robustness. Though GraphNAS shows impressive results on clean graphs, its vulnerability under adversarial attacks is also obvious, e.g. GraphNAS’s performance under adversarial attacks is even worse than that of classic GNNs sometimes. Thus, it’s necessary to make Graph NAS more robust. In CiteSeer, the performance for G-RNA under 20% and 25% structural perturbations are almost twice that of GraphNAS, which shows the strong defense ability of our method.

**Ablation study on robustness metric.** We add an ablation study by removing the robustness metric (denoted as G-RNA w/o rob). Our full G-RNA shows better performance compared to G-RNA w/o rob on all datasets under attacks, demonstrating the effectiveness of our proposed robustness metric.

![Figure 3. Comparison with GCN-NoGraph. G-RNA is the only method that consistently outperforms GCN-NoGraph.](image-url)

**Comparison with not using structures.** When the graph structure is heavily poisoned, aggregating neighbor messages by the message-passing paradigm may not be reliable. As a result, we compare our method with GCNs not using edges, named GCN-NoGraph [20], i.e., a two-layer MLP on node features. Notice that the performance exceeding GCN-NoGraph indicates an effective graph structure, while that below GCN-NoGraph implies an informationless or confusing graph structure for GNNs. Fig. 3 shows the comparison with GCN-NoGraph. For all comparing methods, only the performance of G-RNA always outperforms GCN-NoGraph. GCN-Jaccard performs well for small perturbations but fails to defend against large perturbations. Also, we could see an obvious drop for GCN and GraphNAS when the perturb rate goes beyond 15% for both datasets.

5.2. Understand Operation Robustness

For understanding the robustness of various GNN components, we randomly sample 300 architectures and evaluate
Table 3. The results of node classification accuracy (mean±std, in percentages) under non-targeted attacks (Mettack). **Bold** numbers indicate the best performance. “-” indicates the result is unavailable due to the high time complexity of the model.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Model</th>
<th>Proportion of changed edges (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Cora</td>
<td>Vanilla GNN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GCN</td>
<td>84.28±0.25</td>
</tr>
<tr>
<td></td>
<td>GCN-JK</td>
<td>84.38±0.38</td>
</tr>
<tr>
<td></td>
<td>GAT</td>
<td>84.30±0.65</td>
</tr>
<tr>
<td></td>
<td>GAT-JK</td>
<td>84.16±0.36</td>
</tr>
<tr>
<td></td>
<td>Robust GNN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RGCN</td>
<td>84.60±0.37</td>
</tr>
<tr>
<td></td>
<td>GCN-Jaccard</td>
<td>83.64±0.76</td>
</tr>
<tr>
<td></td>
<td>Pro-GNN</td>
<td>84.64±0.59</td>
</tr>
<tr>
<td></td>
<td>DropEdge</td>
<td>83.35±1.23</td>
</tr>
<tr>
<td></td>
<td>PTDNet</td>
<td>83.70±0.43</td>
</tr>
<tr>
<td>CiteSeer</td>
<td>Vanilla GNN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GCN</td>
<td>72.35±0.49</td>
</tr>
<tr>
<td></td>
<td>GCN-JK</td>
<td>72.14±0.36</td>
</tr>
<tr>
<td></td>
<td>GAT</td>
<td>71.75±0.71</td>
</tr>
<tr>
<td></td>
<td>GAT-JK</td>
<td>71.52±0.90</td>
</tr>
<tr>
<td></td>
<td>Robust GNN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RGCN</td>
<td>72.43±0.41</td>
</tr>
<tr>
<td></td>
<td>GCN-Jaccard</td>
<td>71.03±0.45</td>
</tr>
<tr>
<td></td>
<td>Pro-GNN</td>
<td>71.74±0.76</td>
</tr>
<tr>
<td></td>
<td>DropEdge</td>
<td>71.84±0.21</td>
</tr>
<tr>
<td></td>
<td>PTDNet</td>
<td>72.87±0.46</td>
</tr>
<tr>
<td>PubMed</td>
<td>Vanilla GNN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GCN</td>
<td>86.35±0.15</td>
</tr>
<tr>
<td></td>
<td>GCN-JK</td>
<td>87.07±0.12</td>
</tr>
<tr>
<td></td>
<td>GAT</td>
<td>85.28±0.20</td>
</tr>
<tr>
<td></td>
<td>GAT-JK</td>
<td>85.72±0.14</td>
</tr>
<tr>
<td></td>
<td>Robust GNN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RGCN</td>
<td>86.64±0.08</td>
</tr>
<tr>
<td></td>
<td>GCN-Jaccard</td>
<td>87.11±0.04</td>
</tr>
<tr>
<td></td>
<td>Pro-GNN</td>
<td>83.87±0.24</td>
</tr>
<tr>
<td></td>
<td>PTDNet</td>
<td>83.93±0.10</td>
</tr>
<tr>
<td></td>
<td>Graph NAS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GraphNAS</td>
<td>87.26±0.04</td>
</tr>
<tr>
<td></td>
<td>GASSO</td>
<td>86.27±0.12</td>
</tr>
<tr>
<td></td>
<td>G-RNA w/o rob</td>
<td>87.18±0.07</td>
</tr>
<tr>
<td></td>
<td>G-RNA</td>
<td>87.48±0.12</td>
</tr>
</tbody>
</table>

their performance on clean and attacked graphs. To make a fair comparison, we keep the same training configurations for all selected architectures. Specifically, we train each architecture for 200 epochs with a learning rate of 0.005 and a weight decay of 5e-4. We use the decrease in classification accuracy as the measurement and report the results on the Cora dataset under Mettack (5% perturbation rate), while other datasets show similar trends. The less the performance of one architecture decreases, the more robust that architecture is. Fig. 4a and Fig. 4b show boxplots for the robustness under various architectural designs. For each subplot, the top figure displays the relationship between the operation choice and model test accuracy on the clean graph, while the bottom one shows the operation robustness via accuracy decreases. We make the following observations.

**Intra Message-Passing Layer Design.** Fig. 4a shows the architecture’s robustness for different intra-layer architecture designs. For simplicity, we only consider intra-layer operations for the first layer. We could see that architectural design plays a significant role in both architectural accuracy.
and robustness. The use of pre-processing graph structure mask operations increases GNN models’ robustness but sacrifices some accuracy. For the Cora dataset, NFS is the most effective operation for pre-processing perturbed graph data. GCN is the most fragile correlation coefficient under Mettacl. A plausible reason is that Mettacl adopts a two-layer GCN model as the surrogate model to generate adversarial samples. For neighbor aggregation operations, Sum shows high accuracy and relatively robust performance. What’s more, Max is also able to generate some robust architectures. Besides, combination functions are essential to enhance the robustness of GNNs as we could see a smaller accuracy decrease when using GIN or SAGE operations. Consequently, it is necessary to distinguish self- and neighbor-messages in the message-passing.

**Inter Message-Passing Layer Design.** Our inter-layer designs include skip connections and layer aggregation operations. We also study how the number of message-passing layers affects the robustness of GNNs. The results are summarized in Fig. 4b. Skip connections help elevate both model accuracy and robustness. For the layer aggregation, there are no obvious differences, and LSTM behaves slightly more robustly than the other two operations. More interestingly, increasing the model depth makes GNNs more fragile under adversarial attacks. A 2-layer GNN shows reasonably good performance for the Cora dataset.

**6. Conclusion and Limitations**

In this paper, we propose the first adversarially robust NAS framework for GNNs. We incorporate graph structure mask operations into the search space to enhance the defensive ability of GNNs. We also define a robustness metric that could effectively measure the architecture’s robustness. Experimental results demonstrate the effectiveness of our proposed approaches under adversarial attacks. We empirically analyze the architectural robustness for different operations, which provides insights into the robustness mechanism behind GNN architectures.

Meanwhile, the lack of efficiency is a shared issue for many NAS methods. Since this weakness is not our main focus, we would like to leave this limitation as a future work. Societal impacts are discussed in the Appendix.

**Acknowledgements.**

This work was supported in part by the National Key Research and Development Program of China No. 2020AAA0106300, National Natural Science Foundation of China (No. 62250008, 62222209, 62102222, 61936011, 62206149), Beijing National Research Center for Information Science and Technology (BNRist) under Grant No. BNR2023RC01003, Beijing Key Lab of Networked Multimedia, China National Postdoctoral Program for Innovative Talents No. BX20220185, and China Postdoctoral Science Foundation No. 2022M711813.
References


[3] Heng Chang, Yu Rong, Tingyang Xu, Wenbing Huang, Honglei Zhang, Peng Cui, Xin Wang, Wenwu Zhu, and Junzhou Huang. Adversarial attack framework on graph embedding models with limited knowledge. IEEE Transactions on Knowledge and Data Engineering (TKDE), 2022. 2


[37] Huan Zhao, Quanming Yao, and Weiwei Tu. Search to aggregate neighborhood for graph neural network. In ICDE, 2021. 1, 3, 5


