EDoG: Adversarial Edge Detection For Graph Neural Networks

Xiaojun Xu[1] Hanzhang Wang[2] Alok Lal[2] Carl A. Gunter[1] Bo Li[1]
[1]University of Illinois at Urbana-Champaign [2]eBay
{xiaojun3, cgunter, lbo}@illinois.edu
{hanzwang, allal}@ebay.com

***Abstract*—Graph Neural Networks (GNNs) have been widely applied to different tasks such as bioinformatics, drug design, and social networks. However, recent studies have shown that GNNs are vulnerable to adversarial attacks which aim to mislead the node (or subgraph) classification prediction by adding subtle perturbations. In particular, several attacks against GNNs have been proposed by adding/deleting a small amount of edges, which have caused serious security concerns. Detecting these attacks is challenging due to the small magnitude of perturbation and the discrete nature of graph data. In this paper, we propose a general adversarial edge detection pipeline EDoG without requiring knowledge of the attack strategies based on graph generation. Specifically, we propose a novel graph generation approach combined with link prediction to detect suspicious adversarial edges. To effectively train the graph generative model, we sample several sub-graphs from the given graph data. We show that since the number of adversarial edges is usually low in practice, with low probability the sampled sub-graphs will contain adversarial edges based on the union bound. In addition, considering the strong attacks which perturb a large number of edges, we propose a set of novel features to perform outlier detection as the preprocessing for our detection. Extensive experimental results on three real-world graph datasets including a private transaction rule dataset from a major company and two types of synthetic graphs with controlled properties (e.g., Erdos-Renyi and scale-free graphs) show that EDoG can achieve above 0.8 AUC against four state-of-the-art unseen attack strategies without requiring any knowledge about the attack type (e.g., degree of the target victim node); and around 0.85 with knowledge of the attack type. EDoG significantly outperforms traditional malicious edge detection baselines. We also show that an adaptive attack with full knowledge of our detection pipeline is difficult to bypass it. Our results shed light on several principles to improve the robustness of GNNs.**

I. Introduction

Graph neural networks (GNNs) have been widely applied in many real-world tasks, such as drug screening [7], [11], protein structure prediction [19], and social network analysis [32]. However, recent studies show that GNNs are vulnerable to adversarial manipulation, where carefully crafted instances are able to mislead machine learning models to make an arbitrarily incorrect prediction. Such vulnerabilities have raised great concerns when applying GNNs to security-critical applications. In particular, different types of attacks targeting on GNNs by adding/deleting a small amount of edges within a target graph have been proposed to fool the node classification or subgraph classification tasks [8], [53], [54]. These attacks are shown to be possible in real world scenarios.

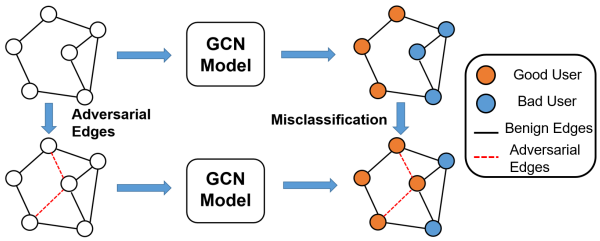


Fig. 1: An example of adversarial attack on graph neural networks (GNNs).

For example, as shown in Figure 1, a malicious user can bypass the malicious access detection system by linking himself with other legitimate users [25].

Detecting such adversarial attacks on GNNs involves several challenges. First, such adversarial attacks focus on local graph properties and aim to create "unnoticeable" perturbations. Therefore, the manipulation of a small number of local edges is not obvious enough to be detected by traditional Sybil detection methods [3]. Second, existing defense/detection methods against adversarial behaviors on machine learning models are not easy to be applied to detect malicious attacks on GNNs for several reasons. For instance, *Robust generative models* are proposed to mitigate adversarial perturbation via denoising autoencoders and Generative Adversarial Networks (GAN) respectively [17], [27], [34]; however, subtle adversarial perturbation in graph-structured data is hard to remove directly through generative models due to the discrete nature of graph data. Third, the perturbations on a graph will show *diverse* behaviors due to different factors, which makes it impossible to learn unified rules to identify them, and in-depth understanding of such adversarial behaviors is required. For instance, adding adversarial edges to a node of higher degree will induce different adversarial patterns (adversarial edges in this case are more likely to be outliers) compared with adding them to a node of low degree.

Given these challenges, in this paper we propose a general detection pipeline EDoG as shown in Figure 2 to detect different unseen adversarial attacks in GNNs. In particular, we propose several detection strategies as components of EDoG, including Link Prediction based method (LP), Graph Generation based method (GGD), and Outlier Detection based method (OD). Here we mainly consider four types of attacks:

(1) adding only one adversarial edge [8]; (2) the number of added adversarial edges is allowed to be up to the degree of a chosen target node, while these adversarial edges are all directly connected to the target node [53]; (3) the number of added adversarial edges is allowed to be up to the degree of a chosen target node, while they are not directly connected to the target node [53];(4) the number of added adversarial edges is allowed to be up to a certain percentage (e.g., 5%) of the number of the edges in the original graph [54].

We found that adversarial edges in the first type of attacks can be characterized by graph generative models which are trained with a large number of subsampled graphs. Therefore, we propose a novel graph generative model based detection method GGD to detect such adversarial edges. In particular, we first sample a number of subgraphs for training the generative model; and since the original graph contains a small number of adversarial edges, based on the *union bound* these sub-graphs will not contain malicious edges with high probability. Such generative models can learn useful patterns from subgraphs and detect malicious edges. In addition, we propose to use link prediction models to filter out suspicious edges coarsely to ensure that the generative models are trained on "clean" edges.

However, when a larger number of adversarial edges are allowed (i.e., the other three types of attacks), the sampled sub-graphs may contain more adversarial edges and therefore the trained graph generative model and link prediction model are not accurate enough to distinguish the added adversarial edges. In such scenarios, we found that the added adversarial edges are more likely to appear as "outliers," while the single adversarial edge would not. As a result, we propose a list of intrinsic features that can be leveraged to perform outlier detection (OD), together with GGD and LP.

By leveraging the intrinsic tradeoff between the stealthiness (hard to appear as "outliers") and sparsity (hard to be sampled in subgraphs) of attacks, the proposed EDoG is able to effectively identify adversarial edges effectively based on at least one of these properties. We conduct extensive experiments on three real-world graph datasets including one private transaction rule graph from a major company as well as two types of synthetic graphs (e.g., Erdos-Renyi and scale-free graphs) with controlled properties. We show that OD can achieve detection AUC above 0.9 for the cases when the target nodes have high degree (larger than 10), which means that if we have knowledge about the attack type it is possible to achieve nearly perfect detection rate. Even without the knowledge of attack types, our results show that the proposed general pipeline EDoG outperforms other state-of-the-art adversarial edge detection methods [31], [33] significantly. In addition, we also evaluate an adaptive attack in which the attacker has full knowledge of our detection pipeline and intentionally aims to bypass it during the attack. We show that the adaptive attack success rate remains very low given our detection pipeline. In summary, we make the following contributions:

- We propose a general light-weighted adversarial edge detection pipeline EDoG on GNNs against the state-of-the-art GNNs based attacks. In particular, we evaluate EDoG to detect against four adversarial attacks on GNNs without requiring knowledge of the attack strategies.
- We propose a novel graph generative model and a filter-and-sample framework to train the graph generative model for adversarial edge detection purpose.
- Based on several interesting observations, we provide a set of effective features on graphs that can be leveraged to perform outlier detection against unseen adversarial attacks as preprocessing.
- We conduct extensive experiments on Cora, Citeseer, transaction rule graph, and synthetic data with controlled properties to detect the state-of-the-art attacks, demonstrating the effectiveness of the proposed pipeline. The detection AUC of EDoG can reach 0.8 without any knowledge of attack types and over 0.85 when we know the attack type. In both scenarios, the proposed EDoG approach outperforms other baseline methods.
- We evaluate EDoG against a strong adaptive attack and show that it is difficult to bypass our detection pipeline.

II. Background

In this section, we will introduce background on neural networks, GNNs and the adversarial attacks on GNNs.

A "c-way classification task" in machine learning is a problem that given an input x and c classes, the model is required to predict which class the input belongs to. In order to deal with a c-way classification task, the output of a neural network is $p \in \mathbb{R}^c$ where the i-th value in p corresponds to the probability that the input belongs to class i. If we know the ground truth class y, we can evaluate the prediction by calculating the cross entropy loss between them:

$$L(p, y) = -\log(p_y)$$

In order to generate a predicted class for the input, the model will take the class with the largest probability:

$$\hat{y} = \arg\max_{y_0} \ p_{y_0}$$

A. Graph Neural Networks

Graph Neural Networks (GNNs) are a class of nerual networks which processes graph data G. In general, $G = (V, E, X)$, where $V = \{v_1, v_2, \ldots\}$ denotes the set of nodes, $E = \{e_1, e_2, \ldots\}$, $e_i \in V \times V$ denotes the set of edges and $X = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{|V|})$ represents the feature vector of each node. Graph data is different from traditional machine learning data in that in addition to the node features, each node has relationship with its neighborhood indicated by the edges. Therefore, a GNN model is usually designed such that each layer will consider the information in the neighbourhood of each node. In particular, a GNN based model calculates an embedding vector $\boldsymbol{\theta}_u$ of each node $u \in V$ via iteratively aggregating information of itself and its neighbours:

$$\boldsymbol{\theta}_u^{(k)} = f_k\big(\boldsymbol{x}_u, \boldsymbol{\theta}_u^{(k-1)}, \{\boldsymbol{x}_v, \boldsymbol{\theta}_v^{(k-1)}\}_{v \in \mathcal{N}(u)}\big),$$

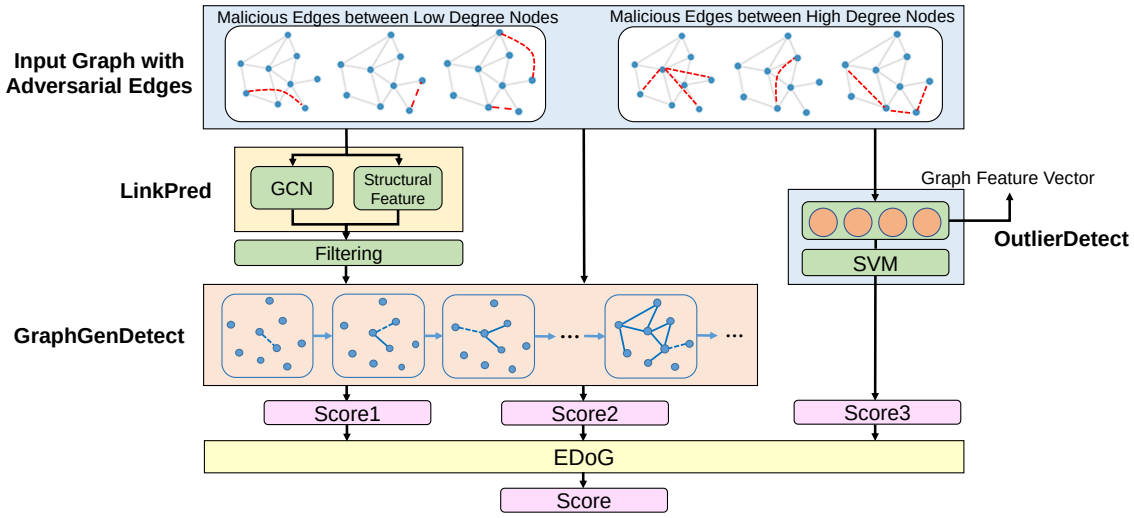


Fig. 2: Illustration of the proposed general detection pipeline EDoG.

where $\mathcal{N}(u)$ denotes the neighbours of u in the graph. The node features $\boldsymbol{x}_u$ serves as the initial embedding $\boldsymbol{\theta}_u^{(0)}$. After we calculated the node embedding, we can use it in different tasks such as node classification and abnormal edge detection.

Many GNN models have been proposed such as Graph Convolutional Networks (GCN) [23] and Structure2Vec [7] with impressive performance on various tasks. In this paper, we will focus on the GCN model. Let $\Theta^{(k)} = (\boldsymbol{\theta}_1^{(k)}, \boldsymbol{\theta}_2^{(k)}, \ldots, \boldsymbol{\theta}_{|V|}^{(k)})$ be the matrix of all node embedding vectors at step k. For GCN, the aggregation function is calculated as:

$$\Theta^{(k)} = \sigma\big(\hat{A}\Theta^{(k-1)}W^{(k)}\big)$$
$$\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$$

where $\tilde{A} = A + I_N$, such that A is the adjacency matrix and I_N the identity matrix. $\tilde{D}$ is a diagonal matrix such that $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. The function σ is a non-linear activation function, and $W^{(k)}$ is the trainable parameters at the k-th layer.

Node classification In a node classification task over graph data, each node v_i has a label $y_i \in \mathcal{Y}$, but we only have access to a small subset of the true labels, i.e., $L_{train} : V_{train} \to \mathcal{Y}$ where $V_{train} = \{v_{i_0}, v_{i_1}, \ldots\} \subset V$ is the set of nodes for which we know the true labels. We also have a set of nodes V_{infer} with $V_{train} \cap V_{infer} = \emptyset$. Given L_{train}, we would like to infer the labels of V_{infer}. That is to say, we seek for a model to give a prediction $\hat{y}_i \in \mathcal{Y}$ to each of the node v_i, and we would like to maximize the classification accuracy, i.e., $\frac{1}{|V_{infer}|}\sum_{v_i \in V_{infer}} \mathbf{1}\{\hat{y}_i = y_i\}$.

When performing the node classification task, a GNN model f first calculates the embedding $\boldsymbol{\theta}_u$ for each node in graph G, which will then be used to calculate the probability vector

$$\boldsymbol{p}_u = f(G, u) = \text{softmax}(W^{out}\boldsymbol{\theta}_u)$$

indicating the probability of each class that node u belongs to. During the training process, the goal is to minimize the cross-entropy loss for the prediction of nodes in V_{train}. During the evaluation process, the predicted class of each node u is given by $\hat{y}_u = \arg\max_y(\boldsymbol{p}_u)_y$.

B. Adversarial Attacks on Graph-structured Data

Recently, several studies [8], [53], [54] have shown that graph neural networks for node classification are vulnerable under adversarial attack. This means that a malicious attacker can modify the graph G subtly into G' before it is fed into the graph neural network such that the graph neural network will generate wrong classification results as desired by the attacker. [8], [53] transform this setting into an optimization problem:

$$\max_{G'} \quad L(f(G', v_t), y_t)$$
$$\text{s.t.} \quad \mathcal{I}(G, G', v_t) = 1$$

where $L(\cdot,\cdot)$ is the cross-entropy loss function between prediction vector of a node v_t and its ground truth class y_t. $\mathcal{I}(G, G', v_t)$ is an equivalency indicator which judges whether the small modification between G' and G is reasonable. This indicator function may vary under different attack setting.

Note that this optimization goal cannot be solved by gradient-based approach because the constraint space is discrete - the value in the adjacency matrix of a graph can only be 0 or 1. In order to solve this problem, [8] defines the equivalency indicator such that the attacker is only allowed to add/delete one edge. The authors propose to parametrize the perturbation generator $G' = h(G, v_t)$ as a neural network and trains it with reinforcement learning [42]. In [53] the equivalency indicator is defined such that the total amount of changed edges and node features is bounded. They propose a simple approximate model of GNNs on which they can analytically solve the optimization problem.

In [54], the optimization problem is defined differently so that it is not restricted to a single node, but over a set of nodes (usually the entire graph):

$$\max_{G'} \quad \sum_{(v,y)\in V_{atk}} L(f(G', v), y)$$
$$\text{s.t.} \quad \mathcal{I}(G, G', V_{atk}) = 1$$

They solve the optimization problem by approximating the gradient of the adjacency matrix of G' so that traditional gradient-based approach is employed.

III. Threat Model and Detection Goal

In this section, we will first introduce the threat model on graph neural networks. Then we will discuss our goal as detecting adversarial attacks against GNNs.

A. Attack Model

Here we mainly focus on the node classification task, where GNNs are used to perform node classification on graph $G = (V, E, X)$. Based on the Kerckhoffs's theory [37], we consider the strongest attacker who has whitebox access to the trained GNNs including the model architecture and parameters. The attacker aims to perform evasion attacks given a trained GNNs, and the strong attack assumption allows us to best evaluate the detection ability of EDoG. **The attacker's goal** is to change the predicted label of a *target node* v_t from the ground truth y_t to the adversarial target $\hat{y'}_t$ ($\hat{y'}_t \neq y_t$) by manipulating the input test graph data in a subtle.

There are mainly two categories of attacks on GNNs:

- Feature attack: The attacker makes small modification to the feature vectors of nodes on the graph. In this attack the graph structure remains unchanged, i.e. $G' = (V, E, X')$.
- Structure attack: The attacker adds or delete a small number of edges in the graph. In this attack the node feature remains unchanged, i.e. $G' = (V, E', X)$.

An attacker may use either or both of the above approaches to attack GNNs. The feature attack is similar to the attack over other continuous data such as computer vision [18] where gradient-based optimization techniques can be used. Many works have been conducted against such kind of attacks. The structure attack, on the other hand, is a newly-proposed attack over discrete data structure. Hence, we mainly focus on the structure attack in this paper and assume that the node features are not changed.

Without loss of generality, here we mainly focus on attackers that add different numbers of malicious edges to the graph, which means $E \subset E'$. There are two reasons for such setting. First, adding edges is usually a cheaper and more practical attack approach than deleting edges. For instance, in an undirected citation network an author can easily add edge by citing others in their own paper but cannot delete edges if their paper has been cited by others. Second, we empirically find that adding edges will yield higher attack success rate than deleting for attackers. As a result, adding edge attack is a more severe threat for learning tasks on graph structured data which we will mainly focus on. On the other hand, we can also leverage the inverse graph to analyze the attacks for deleting edges.

As for the structure attack, we consider the state-of-the-art attack strategies targeting on GNNs [8], [53]. Based on the number of allowed malicious edges and whether these malicious edges are directly connected to the target node of an attacker so as to make direct impact, we can categorize the attacks into three types as below.

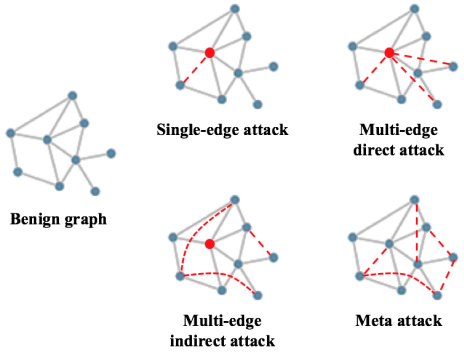


Fig. 3: Examples of four state-of-the-art of attacks we considered in this paper. The red node refers to the target node in the attack and the red dashed lines are the maliciously added edges which we aim to detect.

1) **Single-edge attack**. This attack is proposed by [8] where an attacker is allowed to add only one malicious edge to the graph to perform stealthy attack, i.e. $|E'\backslash E| = 1$.
2) **Multi-edge direct attack**. This attack is proposed by [53] where an attacker is allowed to add several edges to the graph. The maliciously added edges would be connected to the target node and the number of malicious edges should not exceed the degree of the target node. That is, $|E'\backslash E| \leq \deg(v_t)$ and for any edge $\forall e(i,j) \in E'\backslash E$, $v_t \in e(i,j)$.
3) **Multi-edge indirect attack**. This attack is similar to the Multi-edge direct attack except that the added malicious edges are not directly connected to the target node. That is, $|E'\backslash E| \leq \deg(v_t)$ and for edge $\forall e(i,j) \in E'\backslash E$, $v_t \notin e(i,j)$.

In addition, [54] proposes another structure attack which is known as meta attack. In meta attack, the attacker does not have a target node but focuses on the entire graph. The attacker's goal is to make the GNN model to give wrong classification result on as much nodes as possible. They allow the number of added edges to be up to 5% of the total number of existing edges in the graph. Hence, we have the fourth type of attack considered in the paper:

4) **Meta attack**. Here the attacker can add at most 5% of malicious edges, i.e. $|E'\backslash E| \leq 5\% \times |E|$.

Examples of the four types of attacks are shown in Figure 3.

B. Goals of Adversarial Edge Detection

In order to deal with the aforementioned threat, we aim to propose a general pipeline to detect the maliciously added edges $|E'\backslash E|$. Our **detection goal** is as follows. Suppose the defender is provided with a graph $G' = (V, E', X)$. The defender knows that some of the edges may be added maliciously by the attacker as above, and he/she does not have any other information about the attack (e.g. the degree of target nodes or the attack strategies). The goal is to determine

which edges in E' are likely to be malicious. On the high level, the defender will calculate a score s_j for each edge $e_j \in E'$ indicating how likely the edge may be a malicious one. After calculating this score for chosen edges, he/she can either identify adversarial edges directly (by setting a proper threshold) or set priorities for further inspection of "suspicious" edges.

In particular, we would like the detection pipeline to satisfy the following properties:

- The defender only sees the modified graph G' and does not have information about the original graph G. Otherwise the task would become trivial.
- The pipeline should work without any information about the attack, such as the attack strategies or the target node. We will also show later that with such auxiliary information the pipeline may achieve a better performance as ablation studies for understanding purpose.
- The detection pipeline should be general against different attacks. EDoG should be able to generalize and detect malicious edges for 1) different types of graphs, 2) various of attack algorithms and 3) different target nodes with different degrees.

IV. Analysis of Adversarial Attacks on Graphs

In this section, we will first summarize the findings and insights about adversarial edge properties of different types of attacks. We then provide the high level overview and intuitions of our proposed detection approaches, as well as the final detection pipeline EDoG.

A. Malicious Links between Low-degree Nodes: Link Prediction and Generative Model

Here we consider the attack model where the attacker only adds a small number of adversarial edges, which is a sufficient condition for target nodes with low degree. This is because for all the attacks we considered, the maximum number of allowed adversarial edges is up to the degree of target node. When an attacker only adds a very small number of edges to ensure the perturbation is "unnoticeable", we will first assume that the perturbation of malicious edges is small enough to be neglected—any algorithm, except for the target GNN based models, applied over the malicious graph G' will return approximately the same result as if it is applied over G. Thus, an intuitive approach is that we can train a link prediction model using G'. Ideally, the link prediction model would behave as if it were trained using G, so it will predict the node pairs with edges as high scores and the pairs without edges as low scores. The malicious edges, however, do not exist in the benign graph. So the scores of them will be low. Thus, after applying the link prediction algorithm we can check the scores for the edges in G'. Those with low scores should be considered likely to be malicious. In this link prediction based approach, selecting features that would focus more on global structures are more helpful, and therefore we propose to use the GCN based structural features for prediction. We will discuss the used features in detail in the next section.

In addition, in order to better capture the global structure information of the original graph data, we also propose generative models to better approximate the original link distribution. A key step for training generative models is to ensure that the data are strictly *clean* (not malicious) to avoid being "poisoned". One approach for reducing the effect of malicious edges would be to sample subgraphs from the large graph, as illustrated in Figure 4. The intuition is that if we randomly sample many small sub-graphs from a large graph, each edge will only appear in a small proportion of the subgraphs with high probability by union bound. Thus, most subgraphs will contain no malicious edges while preserving the information of the original graph. For example, in our experiment we find that if we sample the graphs by extracting the two-hop neighbour for each node, it turns out that the more than 99% of subgraphs do not contain malicious edges in single-edge attack on average on the dataset we use, and more than 90% for multi-edge attack and meta-attack. Therefore, we can train generative models over the subgraphs to learn a good distribution approximation of the original graph, and then leverage this to detect abnormal edges. Note that the naive link prediction algorithm is not suitable for training over subgraphs in two aspects: first, many link prediction algorithms are based on feature extraction over the entire graph; second, link prediction algorithms tend to have relatively small model capacity to capture the pattern of entire sub-graphs. Therefore, we can only train the proposed graph generative models based on deep neural networks on subgraphs. Based on the generative models, we will be able to identify which edges are the least likely to be generated and these edges are highly likely to be malicious. We find that such generative models are good at discovering patterns from subgraphs and thus detect malicious edges.

B. Malicious Links between High-degree Nodes: Outliers

It turns out that both link prediction and graph generation approach may sometimes fail when applied to multi-edge direct attack, especially when the node degree is large. We attribute this to a principle of the *collective power of malicious edges* which can be understood as: when there are many malicious edges connecting to one node, they confirm the legitimacy of each other mutually. We show an example as in Figure 5. Suppose one node is originally connected to three nodes in class 1. If an attacker adds just one malicious edge that connects it to a node in class 2, this edge will seem abnormal and is easily detected. However, if the attacker instead adds three malicious edges in class 2, the legitimacy of each malicious edge will be supported by the rest, and they will all be judged to be benign.

Under this circumstance, the neighbours of the target node in G' should contain a number of different classes (e.g. 50% in class 1 and 50% in class 2), while the classes of other nodes' neighbors are usually quite uniform. As a result, we may calculate several edge features indicating the information of the neighbourhood of edges, e.g. number of different classes appearing in the neighbourhood of the edge. We could expect

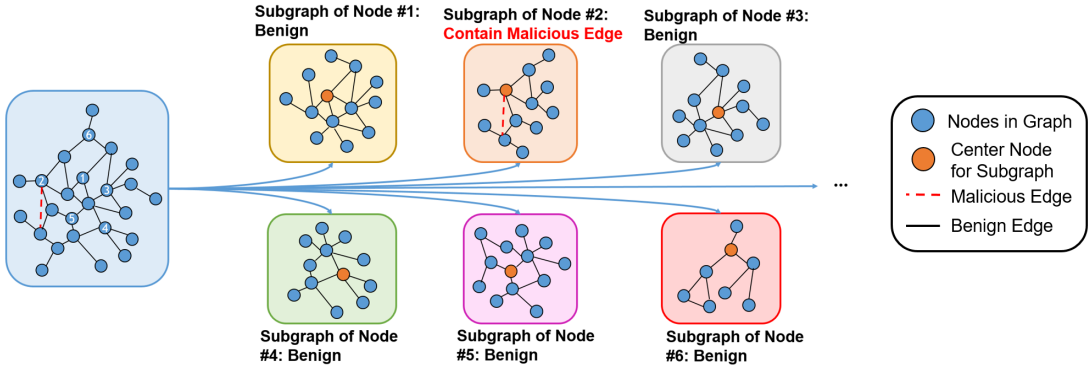
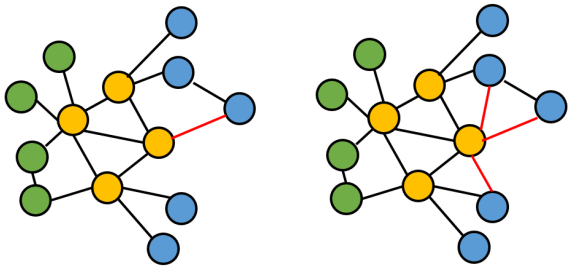


Fig. 4: An illustration of sampling subgraphs to train the graph generative model to capture benign graph structure properties. It is shown that most sampled graphs will not contain malicious edges when only one adversarial edge is added.



(a) Add one edge. (b) Add three edges.

Fig. 5: An example of of "collective power of malicious edges". When only one malicious edge is added (left), it should be easily detected be the graph generative model; but when more malicious edges are added (right), all the malicious edges will appear to be benign.

that the feature vector of a malicious edge would be different from those of benign edges. Therefore, we could build an outlier detection model over all the edges and consider the outliers as malicious.

V. EDoG: Adversarial Edge Detection

In this section, we will introduce the proposed primitives for adversarial detection considering different scenarios, followed by the proposed general detection pipeline EDoG.

A. LinkPred (LP)

A direct link prediction algorithm is first considered to integrate given its simplicity and effectiveness. For example, in [31] the authors calculate five features (as we will discuss in Section VI-A) for each node pair (u, v) and use the feature vector to judge whether the node pair should be connected. However, we can do more than that—using GNN, we may discover some latent feature space in addition to the features used in link prediction. In practice, we first adopt a two-layer graph convolutional network to calculate the node embedding vector $\boldsymbol{\theta}_u$ for each node. Then for each pair node (u, v) we use a bilinear hidden layer to calculate a hidden feature

$$f_{u,v}^{GNN} = \sigma(\boldsymbol{\theta}_u^\intercal W_{edge} \boldsymbol{\theta}_v)$$

Algorithm 1 Algorithm for predicting how likely each node pair in E_{target} should be linked in the graph $G = (V, E, X)$. f_{gen} in the algorithm refers to the Graph Generation Model which takes a graph as input and output the the score indicating the probability of link for each unlinked node pair in the graph.

procedure EdgeLinkProbs(f_{gen}, G, E_{target})
  $LinkProbs \leftarrow [\,]$
  for e **in** E_{target} **do**
    $LinkProbs[e] \leftarrow [\,]$
  $\pi = permutation(|E|)$
  $E_0 = \emptyset$
  for t **in** $0, 1, \ldots, |E| - 1$ **do**
    for e **in** $E_{target} \backslash E_t$ **do**
      $LinkProb = f_{gen}(V, E_t, X)[e]$
      $LinkProbs[e] \leftarrow LinkProbs[e] + [LinkProb]$
    /* Add one edge in each iteration. */
    $E_{t+1} \leftarrow E_t \cup E[\pi_t]$
  for e **in** E_{target} **do**
    $LinkProbs[e] \leftarrow average(LinkProbs[e])$
  return $LinkProbs$

where $\sigma(\cdot)$ is the sigmoid function. This feature is appended to the feature vector of each node pair to form a 6-dimensional vector and then passed through a logistic regression layer. The entire model is trained end-to-end. The inference process is similar to [31]—we calculate the score for all existing edges, and the edges with low scores are likely to be malicious.

B. GraphGenDetect (GGD)

Following the intuition in IV-A, we propose to train a generative models to capture the complex structure of different subgraphs. There have been several works on graph generative models [5], [10], [38], [50], [51]. However, most of the existing generative models aim to generate as diverse as possible graphs from the training set. On the other hand, our goal is to leverage the generative model to predict which edges are more likely to be generated, which means we hope to preserve the properties of original graph. Hence, we aim

Algorithm 2 Algorithm for using Graph Generation model f_{gen} to detect malicious edges on graph $G' = (V, E', X)$.

procedure MALICIOUSEDGEDETECT(f_{gen}, G')
  / Sample sub-graphs. */*
  $G'_0, G'_1, \ldots, G'_n = Sample(G')$
  $LinkProbs \leftarrow []$
  for e **in** E **do**
    $LinkProbs[e] \leftarrow []$
  / Enumerate through sub-graphs. */*
  for G'_i **in** $G'_0, G'_1, \ldots, G'_{n-1}$ **do**
    $SubLinkP \leftarrow$ EdgeLinkProbs(f_{gen}, G'_i, E)
    for e **in** E'_i **do**
      $LinkProbs \leftarrow LinkProbs + SubLinkP[e]$
  / Probability of being malicious. */*
  $MalProbs \leftarrow []$
  for e **in** E' **do**
    $MalProbs[e] \leftarrow 1 - Average(LinkProbs[e])$
  return $MalProbs$

Algorithm 3 Algorithm for training the Graph Generation model f_{gen} given a graph $G' = (V, E', X)$. $E_{nonexist}$ is a randomly sampled set of node pairs such that no edge exists among each node pair.

procedure MODELTRAININGLOSS($f_{gen}, G', E_{nonexist}$)
  / Sample sub-graphs. */*
  $G'_0, G'_1, \ldots, G'_n = Sample(G')$
  $loss \leftarrow 0$
  $E_{all} = E' \cup E_{nonexist}$
  for G'_i **in** $G'_0, G'_1, \ldots, G'_{n-1}$ **do**
    $SubLinkP \leftarrow$ EdgeLinkProbs(f_{gen}, G'_i, E_{all})
    for e **in** E_{all} **do**
      $label = e$ *is an existing edge ?* $1 : 0$
      $loss \leftarrow loss +$ *cross_ent(label,SubLinkP[e])*
  return loss

to design a generative model that could discover the graph structural pattern and does not need to include much diversity in the output. As a result, we propose a deep graph generation model inspired by **sequence generation** approaches [41] – the model will generate the edges one-by-one to construct the entire graph. We train the generative model based on randomly sampled subgraphs, and apply the trained model to predict which edges in E' are the least likely to be generated. These edges with low generative likelihood would be considered to be malicious.

Our model, denoted as f_{gen}, will take as input a graph $G = (V, E, X)$ and output a probability distribution indicating that if we are going to add one edge into the graph, which node pair is likely to be added. That is to say, the output of f_{gen} is a probability distribution over all the node pairs that have not been connected yet:

$$e^{(t)} \sim P\big[(u, v)\big|(u, v) \notin E\big]$$

Based on this model, we can generate a graph given only node information (V, X) in a step-by-step procedure. In particular, we start from an empty edge set $E^{(0)}$ and gradually add edges to the set. At each time step t, we input $(V, E^{(t-1)}, X)$ into the f_{gen}, sample a new edge using the output probability distribution and add the new edge into the edge set to get $E(t)$.

In practice, we use a GCN with bilinear output layer to calculate the probability. We first apply a two-layer GCN to calculate the embedding vector $\boldsymbol{\theta}_u^{(t-1)}$ for each node u at time $t-1$. Then, for each node pair (u, v), we apply a bilinear function $s_{uv}^{(t-1)} = (\boldsymbol{\theta}_u^{(t-1)})^\intercal W \boldsymbol{\theta}_v^{(t-1)}$ to calculate the score. In order to determine which edge will be generated at the next time step, we take softmax of the scores of all the node pairs which have not been connected to calculate the probability as:

$$P\big[(u, v)\big] = \text{softmax}\big(\big\{s_{uv}\big|(u, v) \notin E^{(t-1)}\big\}\big)$$

After training the generative model f_{gen}, we aim to calculate: *given a graph G and a set of node pairs E_{target}, what is the likelihood ratio that there exists an edge between each node pair in E_{target}?*

The detailed algorithm is shown in Algorithm 1. In particular, given a subgraph G_i we will 1) randomly generate a permutation to get an order for edges. Following this order, we are going to start with an empty edge set and add one edge to the graph at each time step; 2) at each time step t, feed in the current adjacency matrix $A^{(t)}$ and node feature X to model f_{gen} and calculate the scores s_{uv} for the desired node pairs in E_{target}; 3) calculate the final score of an edge as the average of the scores which we calculate in all the time steps.

Using Alg. 1, we can infer which edges are likely to be malicious in graph G' as in Alg. 2. During this inference stage, we will first sample sub-graphs from G' and calculate the probability of links for edges in each sub-graph. Then the score of edges in the original graph can be calculated by taking the average accordingly. A small score means the edge is unlikely to be generated, and therefore it is likely to be malicious.

We will use gradient-based method to minimize a training loss iteratively in order to train the model f_{gen}. At each training step, the process for calculating the training loss is shown in Alg 3. Similar to the inference stage, we first sample sub-graphs from $G' = (V, E', X)$. However, in order to train the model, we not only calculate the probability of the linked node pairs (i.e. edges in sub-graphs) but also calculate the probability for a set of unlinked node pairs $E_{nonexist}$. This set is uniformly sampled from the unlinked node pairs in the graph and we sample a different set in different training step. We let the set size be $|E_{nonexist}| = |E'|$. Hence, the training loss at each time step is a binary cross entropy loss over all the node pairs. The ground truth label is 1 if the node pairs is linked in the original graph and 0 otherwise. After calculating the loss, we can apply the gradient-based optimizer to update the model parameters in f_{gen}.

C. Filtering for GraphGenDetect

The graph generation model indeed has a strong capacity in learning patterns from sub-graphs. However, the existence of malicious graphs may still harm the performance of GraphGenDetect because our graph generation model would treat the pattern of malicious edges as benign and learn it well, leading the algorithm to be unstable. Therefore, we propose an effective *filtering* process, i.e. we can first filter away a proportion of edges in the original graph which seems to be suspicious and train our generation model on the filtered graph. If the malicious edges are indeed filtered away, most of the sampled graphs should be benign during the training process and we can expect that our trained generative model will capture the structure properties of benign subgraphs.

In order to decide which edges are likely to be malicious and should be filtered, we can use algorithms introduced before to evaluate the maliciousness score for each edge. In practice, we find our LinkPred approach a good way to filter away edges, as it is a stable algorithm and could reach acceptable performance to filter away malicious edges in most cases. We denote this approach as LinkPred +GraphGenDetect. That is to say, we first apply LinkPred over G' and to get a score for each edge indicating how likely it is malicious. Then we will remove the top k edges with highest malicious scores, getting $G'_{filter} = (V, E'_{filter}, X)$. Thus, we can train our GraphGenDetect model on G'_{filter} and finally use the trained model to detect the malicious edges over G'.

D. OutlierDetect (OD)

Based on the intuition in Section IV-B, we propose the OutlierDetect to further identify adversarial edges mainly for high-degree victim nodes. The intuition of this approach comes from an observation on the attacker behaviour during direct structure attack: in order to make the target victim node be misclassified into another class (say class 1), the attacker tends to add many edges between the target nodes with nodes that belong to class 1. Hence, the class distribution of the target node's neighbours can be quite diverse considering its previous connections. In contrast, the classes in a benign node's neighbourhood should be quite uniform. Inspired by this phenomenon, we propose a novel outlier detection model for the edges based on the class distribution of the neighbourhood nodes of an edge. In particular, we calculate the following features for each node:

- The number of different classes of the neighbourhood nodes.
- Average appearance time of each class in the neighbourhood nodes.
- Appearance time of the most frequently appeared class in the neighbourhood nodes.
- Appearance time of the second most frequent class in the neighbour (0 if only one class is in the neighbourhood).
- Standard deviation of the appearance time of each class in the neighbourhood.
- Logarithm of the betweenness centrality [14] of each node in the graph.

Note that we do not have ground truth classes information for most nodes, so we would first fit a GNN over the graph G' and use the prediction as the label. For each edge, we calculate the above features for both nodes and concatenate them together, constructing a 10-dimensional feature vector. We then train a one-class SVM with the RBF kernel over these edge feature vectors to detect the outliers. The trained model will calculate a score for each edge indicating its abnormality. The larger the value is, the more likely that the edge is a malicious one.

E. General Pipeline for Detection - EDoG

The primitive approaches we propose above focus on different attacking scenarios. In particular, LinkPred + GraphGenDetect and GraphGenDetect approach work well in most cases, except for the case when many malicious edges are connected to a single node (target node with high degree). On the other hand, OutlierDetect is very good at detecting such kinds of attacks with target nodes of high degree. Therefore, similar to [6], in which a strong model is fit over graph data, we need to design a unified framework for the overall detection on adversarial edges.

Based on the node degree information, our final pipeline, namely Edge Detection of Graph (EDoG), is shown in Figure 2. It is an aggregated model which averages the output of LinkPred + GraphGenDetect, OutlierDetect and GraphGenDetect. In particular, we apply the three approaches over the graph and get their prediction score for each edge. Then for edges between high-degree nodes (in practice we choose the criteria that the sum of degree of the two nodes is larger than 6), we use the average of scores from the three approaches; otherwise, for low degree nodes we only use the average score of LinkPred + GraphGenDetect and GraphGenDetect. The advantage in doing this is: first, GraphGenDetect and LinkPred + GraphGenDetect perform not very well at attacks with high node degree by Multi-edge direct attack, and incorporating it with OutlierDetect significantly improve the detection performance; second, the GraphGenDetect-based approach performs well but sometimes not very stable, so an aggregated model could help improve its robustness significantly.

VI. Experimental results

In this section, we will first introduce the dataset and evaluation metrics we use, followed by the attack models, and performance analysis of the proposed detection methods.

A. Experimental Setup

Datasets and evaluation metrics. We evaluate our detection model on two public citation networks, one private transaction rule graph from a major company, and two types of synthetic dataset with controlled properties. The benign accuracy of our model on the datasets are shown in Appendix A. We hope that such variety of datasets can demonstrate the flexibility of our approach.

The real-world network datasets we use are Cora [26] and Citeseer [15]. These two datasets are citation networks where

a node represents a research paper and an edge represents a citation between two papers. Cora has 2,708 nodes and 5,429 edges; Citeseer has 3,327 nodes and 4,732 edges. Each node contains a bag-of-word feature vector and has a ground truth label indicating which type of paper it is. During the training process, only a small part of node labels are available (140 for Cora and 120 for Citeseer). The model is trained to determine what are the labels for the other nodes. Cora is a 7-way classification task and Citeseer is 6-way. This setting is commonly used in the task of node classification on graph data [8], [23], [53].

We also evaluate EDoG on a private transaction rule graph from a major company. In this rule graph (denoted as Rule), each node represents a rule in the system and each edge represents a relationship that two rules are frequently co-triggered. We extract a subgraph with 719 nodes and 3,375 edges. Each node has a related feature vector, including information such as author ID and its application in the system. The node classification task is to determine whether a rule is a temporary test rule or not.

For experiments on synthetic graphs with controlled properties, we generate two Erdos-Renyi graphs [12] and one scale-free graph [2]. The two Erdos-Renyi graphs are G_{n,p_1} and G_{n,p_2} with $n = 1000$ and $p_1 = \frac{\ln n}{n}$, $p_2 = \frac{2\ln n}{n}$. The scale-free graph is generated using Barabasi-Albert algorithms, with 1,000 nodes and parameter $m = 1$. We would like to assign node features and node labels that are related with the graph structures. Therefore, given a synthetic graph we first assign a 20-dimensional random feature e_u to each node u. Then we let the node features to correlate with its neighbours by repeat:

$$e_u = \sum_{v \in \mathcal{N}(u)} e_v, \qquad e_u = e_u / ||e_u||_2$$

where $\mathcal{N}(u)$ is the neighboring nodes of u. After repeat the process several times (in practice we repeat 3 times), we can get a hidden feature vector e_u which is related with graph structures. The final node feature vector x_u is a discrete random binary vector, the probability that the i-th bit of x_u equals 1 is:

$$Pr[x_u^{(i)} = 1] = sigmoid(e_u^{(i)})$$

And the label of u is $y_u = 1$ if $\sum_i x_u^{(i)} > 0$ and otherwise $y_u = 0$. The node classification accuracy can reach around 80% for ER graphs and around 75% for scale-free graphs.

For each dataset and attack approach, we randomly pick several target victim nodes and perform the state-of-the-art attacks to generate malicious edges. Then we perform the detection method on the new graphs and check whether the malicious edges can be detected without attack information. The evaluation metric is the Area Under ROC Curve (AUC), which is a commonly used metric to verify the performance of a detection method.

Attack strategies. We evaluate the attack strategies as introduced in Section III-A. We only choose the target nodes that are successfully attacked and randomly sample the target nodes with different degrees. We observe in practice that Multi-edges direct attack is the most successful attacking model, followed by Single-edge attack and finally Multi-edges indirect attack. Therefore, we selected 20, 10 and 6 target nodes respectively for these three attack methods on real-world data. For synthetic data, we simply pick two target nodes, one with the smallest degree and the other with the largest degree. The selected target node degrees are shown in the Table I. For the meta-attack, we follow the same setting as in the paper[1] and add 5% malicious edges to the graph. We use a standard 2-layer GCN for the classification tasks and the training setting is the same as in [8], [53], [54].

TABLE I: Degree of the selected target victim nodes. We aim to cover diverse target nodes with a large range of degrees.

	Cora	Citeseer	Rule
Single-edge	1,2,3,4,5, 6,6,7,8,10	1,2,3,3,4, 4,5,6,7,9	1,2,3,4,4, 5,5,6,7,10
Multi-edges direct	1,2,3,4,4,6, 7,8,8,10,12, 14,12,13,15,31, 19,32,16,17	1,2,2,3,4,5, 6,6,7,8,9,10, 11,12,13,15, 16,17,18,20	1,2,3,4,5,6,7, 8,9,10,11,12, 14,14,15,16, 17,18,19,20
Multi-edges indirect	3,4,14,12,13,17	1,4,6,8,13,17	3,6,8,11,11,15

Baseline Detection Approaches. We compare the proposed EDoG with two state-of-the-art detection approaches as below. We will show the performance of other traditional metrics in Appendix B.

1) Anomaly Link Discovery *Approach (*ALD*):* Our first baseline is the anomaly link detection approach as proposed in [31]. This approach trains a link prediction-based algorithm for detection. For each node pair (u, v) it calculates five features, including the similarity of neighbours, the number of common neighbours, the distance between two nodes, the preferential attachment of two nodes and the similarity of the node features of u and v. Then it trains a logistic regression classifier over the feature vectors of the node pairs, where the ground truth label of node pairs with edge is 1, otherwise 0. After the model is trained, we calculate the probability of link for each existing edge. The smaller the probability is, the more likely that edge is a malicious one.

2) Katz Index *Approach (*Katz*):* Another baseline is an anomaly link discovery method described in [33], in which they used a high-order heuristic named Katz index [24] to measure the connectivity of each node pair (u, v) as

$$Katz(u, v) = \sum_{l=1}^{\infty} \beta^l |walks^l(u, v)|$$

where β is damping factor to assign more weight to shorter walks and $walks^l(u, v)$ is the set of random walks with the length of l between u and v. The intuition behind this measure is that compared with the normal links, malicious links often have small Katz index values. Then we can calculate the Katz index for each node pair and used the softmax function for normalization. After that, the value assigned for each node pair is bounded between 0 and 1, and we use this value as the

[1] https://github.com/danielzuegner/gnn-meta-attack

TABLE II: The average AUC of EDoG and baselines against different attacks without knowledge of the attack type.

	Single-edge attack	Multi-edges direct attack	Multi-edges indirect attack	Meta attack
ALD	0.5989	0.4293	0.3499	0.4759
Katz	0.8502	0.6214	0.7985	0.6964
EDoG	**0.8610**	**0.7551**	**0.8288**	**0.7277**

probability for the existence of the link between the nodes. The smaller the probability is, the more likely that edge is a malicious one.

Implementation Details. We implement all the detection models in Pytorch [29] except for the ALD and Katz and OutlierDetect where we use the scikit-learn toolkit [30] for logistic regression and one-class SVM. In order to sample subgraphs from the original graph, we iterate through each node and extract its two-hop neighbourhood as a subgraph. Thus, we can get a set of subgraphs whose cardinality equals the number of nodes in the original graph. For LinkPred, we train it for 500 epochs using SGD optimizer with learning rate of 0.01 and we sample among node pairs which are not connected so that the number of positive and negative labels is the same. For OutlierDetect we fit a one-class svm with radial basis function kernel. For GraphGenDetect we train it using Adam optimizer [22] for 15 epochs with learning rate 0.001. We observe that the detection result of the GraphGenDetect model is reasonable fast (e.g. 5 epochs can be enough) while the result can be non-stable. Therefore, during test time we evaluate the trained model from the 6th to 15th epoch and take the average scores as the final prediction. For filtering model LinkPred + GraphGenDetect, we filter away 50% of the edges using the result of LinkPred.

B. Detection Performance on Real-world Datasets

We will first provide the overall performance comparison between our proposed pipeline EDoG and the state-of-the-art baselines [31], [33] with and without knowledge of the attacker's approach to demonstrate its efficacy, and then present how each of our proposed detection approaches perform on different real-world datasets and attacks strategies.

Overall Performance without knowledge of attack type. Despite the efficacy of our approach compared with baselines, in many cases we do not know the attack approach. Therefore, we need a uniform pipeline, EDoG, to defend against the various types of attacks. The performance of EDoG compared with the baselines is shown in Table II. To avoid data dependency and make the comparison more clear, we average the results over datasets to evaluate the overall performance. As we can see, our approach EDoG outperforms the baselines on all the tasks. Also, we observe that defending against Single-edge attack and Multi-edges indirect attack are relatively easy tasks. Our pipeline can achieve over 0.8 AUC on these tasks. The other two attacks are relatively hard to defend. Some results of baseline approaches are even worse than random guess. As we discussed before, this may be due to the principle of the *collective power of malicious edges*. Nevertheless, our approach can still get an over 0.7 detection AUC against such kind of attack since we use the scores of OutlierDetect to detect edges between high degree nodes. This again demonstrates the robustness of our general pipeline.

Overall Performance with knowledge of attack type. As shown in Table III, our approach EDoG outperforms the state-of-the-art baselines in most cases significantly when there is knowledge about the attack type. The performance of different detection approaches matches with our intuition discussed above, and different models do well against different types of attacks. Hence, when we have knowledge of the attack approach, we can use the detection method that is good at dealing with it. This is also useful when we want to defend against specific type of attacks. In particular, LinkPred + GraphGenDetect works well for Single-edge attack and Multi-edges direct attack on small-degree target nodes (degree ≤ 5). For Multi-edges direct attack on large-degree target nodes, OutlierDetect is the appropriate choice. On Multi-edges indirect attack, it turns out that both GraphGenDetect and LinkPred + GraphGenDetect would work well and we choose LinkPred + GraphGenDetect to be coherent with that in Single-edge attack. For Meta attack, EDoG is used since Meta-attack may contain various types of malicious edges. The average AUC of the proposed approaches is close to 0.85, demonstrating the efficacy of our detection pipeline EDoG against different attacks.

In order to explore the detection performance of different detection primitives, we show the results of each detection method that we proposed in Appendix C. We will observe that graph generation-based method has superior performance for low-degree attacks while outlier detection method performs better on high-degree attacks. With our ensemble method EDoG, the performance will be good for all tasks.

C. Analysis on Synthetic Dataset

Besides the real-world graph dataset, we also evaluate the attack approaches and our detection framework on the synthetic ER graphs and scale-free BA graph, aiming to obtain in-depth analysis of our detection approaches in the controlled environments. In this set of experiments, we perform in-depth analysis for how well the EDoG perform in terms of detecting adversarial edges, if the graph has certain properties to make the detection harder. The result is shown in Table IV. As we can see, the performance of our detection approach over synthetic graphs is even better than that over real-world data. The performance on the scale-free graph, which is similar to real world citation network structure, reaches an average of above 92% AUC on all tasks. The performance on Erdos-Renyi graphs are comparatively low, yet still achieve an average AUC of around 80%. This shows that our approach does exploit the structure of scale-free graph to improve the performance. Also, we observe that our approach beats

TABLE III: AUC of the our detection approaches compared with baselines when we have knowledge of the attack type.

		Single-edge attack		Multi-edges direct attack				Multi-edges indirect attack		Meta attack
	Node degree	$[1, 5]$	$(5, +\text{inf})$	$[1, 5]$	$(5, 10]$	$(10, 15]$	$(15, +\infty)$	$[1, 10]$	$(10, +\text{inf})$	-
Cora	ALD	0.6745	0.4331	0.5363	0.4312	0.4372	0.4302	0.5384	0.4350	0.4820
	Katz	**0.9213**	0.8282	0.7965	0.5727	0.3984	0.3946	0.8262	0.7562	0.6189
	(LP + GGD) ⊗ OD ⊗ EDoG	0.9088	**0.9110**	**0.8319**	**0.7378**	**0.9144**	**0.9123**	**0.8368**	**0.8414**	**0.6884**
Citeseer	ALD	0.7942	0.5806	0.4853	0.2696	0.2726	0.2744	0.4229	0.3715	0.3956
	Katz	0.7677	0.6482	**0.6049**	0.4004	0.2851	0.2490	0.6401	0.6124	0.5042
	(LP + GGD) ⊗ OD ⊗ EDoG	**0.8350**	**0.8750**	0.6047	**0.8024**	**0.9176**	**0.9220**	**0.7827**	**0.6712**	**0.5223**
Rule	ALD	0.5609	0.5499	0.4449	0.4752	0.5357	0.5585	0.2577	0.0741	0.5502
	Katz	0.9689	0.9666	0.9696	0.9389	0.9392	0.9071	0.9796	0.9766	0.9660
	(LP + GGD) ⊗ OD ⊗ EDoG	**0.9845**	**0.9885**	**0.9946**	**0.9865**	**0.9841**	**0.9832**	**0.9934**	**0.9796**	**0.9724**

TABLE IV: The average AUC on synthetic dataset. BA stands for the scale-free graph generated by Barabasi-Albert algorithm. ER is the average result of the Erdos-Renyi graphs generated with $p_1 = \frac{\ln n}{n}$ and $p_2 = \frac{2 \ln n}{n}$ respectively.

	Single-edge attack		Multi-edges direct attack		Multi-edges indirect attack		Meta attack	
Dataset	BA	ER	BA	ER	BA	ER	BA	ER
ALD	0.5000	**0.8465**	0.5000	0.6146	0.7495	0.6535	0.5194	0.4550
Katz	0.6265	0.4298	0.2633	0.3077	0.1667	0.6100	0.6123	0.4322
EDoG	**0.9975**	0.8289	**0.7896**	**0.7397**	**0.9437**	**0.8247**	**0.9626**	**0.7985**

baselines on all Multi-edges direct attack and Meta attack, which we consider as hard tasks. For the other two tasks, the baseline can sometime outperform our approach.

Hence, we conclude that EDoG can generalize to different graph structures. In addition, the baseline approaches can sometimes work well in clean synthetic dataset, but in complicated and noisy real-world graph structure, the robustness of our approach is superior.

D. Visualization of Attack and Detection

To intuitively understand the adversarial attack and corresponding detection performance, we include a visualization of an attack and our detection on the Rule dataset as shown in Fig. 6. The node in the middle is the target node and the attacker performed multi-edges direct attack to inject several malicious edges connected to it. We can observe that the target node is a test rule and the attacker's goal is to fool the model to predict it as non-test rule, so the malicious edges are all connected to the non-test rules. During detection, our EDoG pipeline detects such adversarial edges easily. All the malicious edges are detected with a high EDoG score.

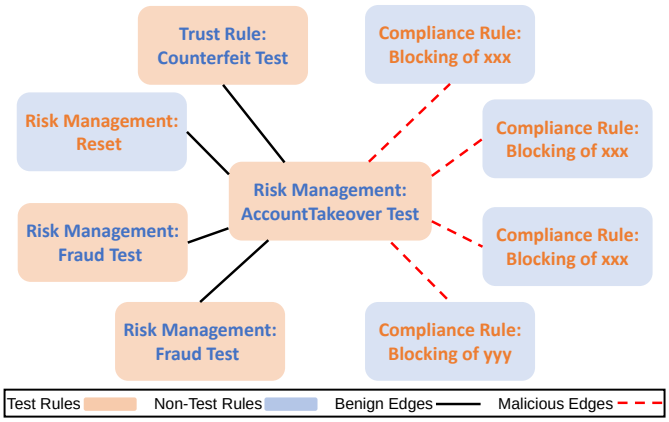


Fig. 6: Visualization of one multi-edges direct attack on Rule dataset. The node in the middle is the target node. Our EDoG pipeline successfully detects all the injected malicious edges.

E. Adaptive Attacks

In this subsection, we consider the adaptive attack where the attacker has full knowledge of our detection pipeline and will intentionally try to bypass our detection during their attack. We design the adaptive attack as follows: given the clean graph G, the attacker will first run the EDoG pipeline and get the prediction scores for all the node pairs. Then, during the attack process, the attacker will only inject the malicious edges with a lower EDoG prediction score (in practice we use only the lower 25% edges). With such process, the attacker tries to only inject the edges that seem benign to the EDoG system. Note that the edges are not guaranteed to evade the detection, since the EDoG score will change after the injection of malicious edges.

We show the results of adaptive attacks as in Table V and compare with the standard attack which is allowed to inject arbitrary edges. In order to perform adaptive attack, the attacker's choice of malicious edges are restricted. Therefore, we can observe that the attack success rate is greatly reduced. In most cases the success rate is reduced by over 50%. This shows that our detection pipeline is difficult to bypass even if the attacker has full knowledge of it.

TABLE V: Comparison of the attack success rate between standard attack and adaptive attack against EDoG.

	Attack	Cora	Citeseer	Rule
Standard Attack	Single-edge	0.475	0.463	0.246
	Multi-edges direct	0.729	0.842	0.556
	Multi-edges indirect	0.149	0.211	0.052
	Meta	0.376	0.333	0.240
	Attack	Cora	Citeseer	Rule
Adaptive Attack	Single-edge	0.070	0.217	0.146
	Multi-edges direct	0.486	0.397	0.483
	Multi-edges indirect	0.087	0.066	0.033
	Meta	0.026	0.022	0.000

F. Detecting Random Edges on Graphs

In order to evaluate whether our pipeline is able to capture the behaviours of pure malicious attackers, we conduct another

experiment where we add random edges into the graph. The detailed results and analysis are shown in Appendix D. We can observe that baseline approaches will consider random edges as some malicious behaviour, while ours will ignore those randomness and focus only on those malicious edges.

VII. Related Work

Adversarial attacks on graphs. Recently the robustness of machine learning models has been studied in numerous settings such as adversarial examples [18], [44], [46], [47]. Most of the researches on on a continuous input space (e.g. images and audios). As for text domain, some of attacks are based on manually constructed perturbations ([20], [35]) and [16] adopted the gradient attacking method to the embedding space of texts. As for graphs, performing attacks can be more difficult especially when we want to make perturbations on edges, since the gradient attacking method is not easy to be used here. So far there are several attacks proposed on graphs: some of the works attempted to attack the graph neural networks [8], [53], [54], as [8] proposed three attack methods on both node classification and graph classification problems and [53] developed an attack method targeting on a particular node based on greedy approximation scheme. Rather than reducing the classification result of a particular node, [54] attacked the overall performance of node classification tasks through meta learning method. Moreover, [4], [40] studied the vulnerability of unsupervised node embedding models.

Defense methods against adversarial attacks. Defense on neural networks is much harder compared with attacks ([27]). Currently, most of the defense methods are based on (1) *changing the architecture of deep learning models:* [28] leveraged distillation training techniques and reduce the magnitude of gradients between the pre-softmax layer (logits) and softmax outputs. [43] used the graph attention network ([45]) to harness information from a graph of peer samples to improve the robustness of network. (2) *adversary training methods:* [18] managed to augment the training dataset with adversarial examples. (3) *data preprocessing:* [49] hardened the deep learning models against adversarial perturbations on images through reducing the color depth and smoothing on spatial domain. [27], [34] attempted to filter the adversarial noise of the input samples with autoencoders and Generative Adversarial Networks (GAN) ([17]) respectively. Although those approaches are somehow effective, they still have some limitations and the task of defending against adversarial attacks is still challenging.

In this paper we focus on detecting malicious edges in a graph under attack. There are also works aiming to directly defend the attacks and mitigate the effect of malicious edges [52]. Comparisong with defense approaches, our detection approach is superior in two properties: first, detecting the malicious edge can help identify the attacker. For example, in the case of social network we can find out which user is adding the malicious link. Second, the defense approaches are usually more prone to adversarial attack, since they incorporate the defense mechanism in the model and are therefore easier to be attacked as a whole when the attacker has knowledge of such mechanism.

Robust Graph Neural Networks. Considering that the threat of adversarial attacks on GNNs is newly emereged, the research with regard to defense methods on graph neural networks is limited. [55] proposes a certificate for GNN robustness under adversarial attack by changing node features. [13] proposes adversarial retraining pipeline to improve GNN model robustness and it also deals with node feature attack. The goal of these two works is different from ours since we are defending against graph structure attack. [48] also proposes an adversarial retraining pipeline to defend against graph structure attack. They evaluate over the meta attack and the retrained model still suffers from performance decrease. [52] proposes a GNN structure where each layer is no longer deterministic but a stochastic transformation layer. They show that using this structure the model performance will increase under adversarial structure attack. However, when the number of added edges increases the model will still fail.

Anomaly detection on graphs. There have been various researches on the topic of '*Anomaly Detection*' or '*Fraud Detection*' on graphs [1], [21], [36]. Note that the detection purpose is different from ours: anomaly detection on graphs aims to find nodes/edges that differ a lot from others. While the two proposed subtle adversarial attack on graphs appear to contain too small magnitude of perturbation to be detected, and here we focus on graph neural networks instead of general graph analysis. In addition, [6] studied on how to fit a good model over graph data when only one graph is available. They focus on the classification performance while our work focuses on the detection of malicious edges, and we have completely no supervision. Our work is going to defend against graphical network models by detecting malicious edges with original approaches. [9], [39] propose virtual adversarial training on GNNs as a way to improve model performance but they do not evaluate their model against adversarial attack.

VIII. Conclusions

Overall, we propose the first general detection pipeline EDoG against the state-of-the-art attack on GNNs. We investigate into the attack and defense properties and find that different attack strategies led to different behaviors: malicious edges connecting low degree nodes will not likely to appear as outliers while the ones connecting high degree nodes will. Thorough experiments show that the average detection AUC of EDoG can reach above 80% and adaptive attacks are hard to be performed given the complexity of the detection pipeline. These results shed light on the design of robust GNNs against attacks on graph-structured data.

Acknowledgements

This work is partially supported by NSF grant No.1910100, NSF CNS No.2046726, a C3.ai DTI Award, and the Alfred P. Sloan Foundation.

REFERENCES

[1] L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: a survey," *Data Mining & Knowledge Discovery*, vol. 29, no. 3, pp. 626–688, 2015.

[2] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Reviews of modern physics*, vol. 74, no. 1, p. 47, 2002.

[3] H. Bansal and M. Misra, "Sybil detection in online social networks (osns)," in *IACC*. IEEE, 2016, pp. 569–576.

[4] A. Bojcheski and S. Günnemann, "Adversarial attacks on node embeddings," *arXiv preprint arXiv:1809.01093*, 2018.

[5] A. Bojchevski, O. Shchur, D. Zügner, and S. Günnemann, "NetGAN: Generating graphs via random walks," in *ICML*, 2018, pp. 610–619.

[6] S. Chen and J.-P. Onnela, "A bootstrap method for goodness of fit and model selection with a single observed network," *arXiv preprint arXiv:1806.11220*, 2018.

[7] H. Dai, B. Dai, and L. Song, "Discriminative embeddings of latent variable models for structured data," in *ICML*, 2016, pp. 2702–2711.

[8] H. Dai, H. Li, T. Tian, X. Huang, L. Wang, J. Zhu, and L. Song, "Adversarial attack on graph structured data," in *ICML*, 2018, pp. 1115–1124.

[9] Z. Deng, Y. Dong, and J. Zhu, "Batch virtual adversarial training for graph convolutional networks," *arXiv preprint arXiv:1902.09192*, 2019.

[10] M. Ding, J. Tang, and J. Zhang, "Semi-supervised learning on graphs with generative adversarial nets," in *CIKM*. ACM, 2018, pp. 913–922.

[11] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *NIPS*, 2015, pp. 2224–2232.

[12] P. Erdős and A. Rényi, "On random graphs," *Publicationes Mathematicae Debrecen*, vol. 6, pp. 290–297, 1959.

[13] F. Feng, X. He, J. Tang, and T.-S. Chua, "Graph adversarial training: Dynamically regularizing based on graph structure," *arXiv preprint arXiv:1902.08226*, 2019.

[14] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, pp. 35–41, 1977.

[15] C. L. Giles, K. D. Bollacker, and S. Lawrence, "Citeseer: An automatic citation indexing system," in *Proceedings of the third ACM conference on Digital libraries*. ACM, 1998, pp. 89–98.

[16] Z. Gong, W. Wang, B. Li, D. Song, and W. S. Ku, "Adversarial texts with gradient methods," *arXiv preprint arXiv:1801.07175*, 2018.

[17] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *NIPS*, 2014, pp. 2672–2680.

[18] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[19] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NIPS*, 2017, pp. 1024–1034.

[20] R. Jia and P. Liang, "Adversarial examples for evaluating reading comprehension systems," in *EMNLP*, Copenhagen, Denmark, September 2017, pp. 2021–2031.

[21] M. Jiang, P. Cui, A. Beutel, C. Faloutsos, and S. Yang, "Catchsync: catching synchronized behavior in large directed graphs," in *SIGKDD*. ACM, 2014, pp. 941–950.

[22] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[23] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.

[24] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," in *CIKM*, 2003, pp. 556–559.

[25] P. K. Manadhata, S. Yadav, P. Rao, and W. Horne, "Detecting malicious domains via graph inference," in *European Symposium on Research in Computer Security*. Springer, 2014, pp. 1–18.

[26] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore, "Automating the construction of internet portals with machine learning," *Information Retrieval*, vol. 3, no. 2, pp. 127–163, 2000.

[27] D. Meng and H. Chen, "Magnet: a two-pronged defense against adversarial examples," in *SIGSAC*. ACM, 2017, pp. 135–147.

[28] N. Papernot, P. Mcdaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *Security and Privacy*, 2016, pp. 582–597.

[29] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS Workshop*, 2017.

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *JMLR*, vol. 12, pp. 2825–2830, 2011.

[31] B. Perozzi, M. Schueppert, J. Saalweachter, and M. Thakur, "When recommendation goes wrong: Anomalous link discovery in recommendation networks," in *SIGKDD*. ACM, 2016, pp. 569–578.

[32] J. Qiu, J. Tang, H. Ma, Y. Dong, K. Wang, and J. Tang, "Deepinf: Social influence prediction with deep learning," in *SIGKDD*. ACM, 2018, pp. 2110–2119.

[33] M. J. Rattigan and D. Jensen, "The case for anomalous link discovery," *Acm Sigkdd Explorations Newsletter*, vol. 7, no. 2, pp. 41–47, 2005.

[34] P. Samangouei, M. Kabkab, and R. Chellappa, "Defense-GAN: Protecting classifiers against adversarial attacks using generative models," in *ICLR*, 2018.

[35] S. Samanta and S. Mehta, "Towards crafting text adversarial samples," *arXiv preprint arXiv:1707.02812*, 2017.

[36] N. Shah, H. Lamba, A. Beutel, and C. Faloutsos, "The many faces of link fraud," in *ICDM*. IEEE, 2017, pp. 1069–1074.

[37] C. E. Shannon, "Communication theory of secrecy systems," *Bell system technical journal*, vol. 28, no. 4, pp. 656–715, 1949.

[38] M. Simonovsky and N. Komodakis, "Graphvae: Towards generation of small graphs using variational autoencoders," *arXiv preprint arXiv:1802.03480*, 2018.

[39] K. Sun, H. Guo, Z. Zhu, and Z. Lin, "Virtual adversarial training on graph convolutional networks in node classification," *arXiv preprint arXiv:1902.11045*, 2019.

[40] M. Sun, J. Tang, H. Li, B. Li, C. Xiao, Y. Chen, and D. Song, "Data poisoning attack against unsupervised node embedding methods," *arXiv preprint arXiv:1810.12881*, 2018.

[41] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *NIPS*, 2014, pp. 3104–3112.

[42] R. S. Sutton *et al.*, *Introduction to reinforcement learning*, vol. 2, no. 4.

[43] J. Svoboda, J. Masci, F. Monti, M. Bronstein, and L. Guibas, "Peernets: Exploiting peer wisdom against adversarial attacks," in *ICLR*, 2019.

[44] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[45] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," *ICLR*, 2018.

[46] C. Xiao, B. Li, J.-Y. Zhu, W. He, M. Liu, and D. Song, "Generating adversarial examples with adversarial networks," in *IJCAI*, 2018.

[47] C. Xiao, J.-Y. Zhu, B. Li, W. He, M. Liu, and D. Song, "Spatially transformed adversarial examples," in *ICLR*, 2018.

[48] K. Xu, H. Chen, S. Liu, P.-Y. Chen, T.-W. Weng, M. Hong, and X. Lin, "Topology attack and defense for graph neural networks: An optimization perspective," *arXiv preprint arXiv:1906.04214*, 2019.

[49] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," *arXiv preprint arXiv:1704.01155*, 2017.

[50] J. You, B. Liu, Z. Ying, V. Pande, and J. Leskovec, "Graph convolutional policy network for goal-directed molecular graph generation," in *NIPS*, 2018, pp. 6412–6422.

[51] J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec, "GraphRNN: Generating realistic graphs with deep auto-regressive models," in *ICML*, 2018, pp. 5708–5717.

[52] D. Zhu, Z. Zhang, P. Cui, and W. Zhu, "Robust graph convolutional networks against adversarial attacks," in *SIGKDD*. ACM, 2019.

[53] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," in *SIGKDD*. ACM, 2018, pp. 2847–2856.

[54] D. Zügner and S. Günnemann, "Adversarial attacks on graph neural networks via meta learning," in *ICLR*, 2019.

[55] ——, "Certifiable robustness and robust training for graph convolutional networks," in *SIGKDD*. ACM, 2019, pp. 246–256.

TABLE VI: The benign accuracy of the model on different datasets.

Task	Cora	Citeseer	Rule
Benign Accuracy	0.819	0.697	0.859

APPENDIX A
BENIGN ACCURACY

In Table VI, we show the benign accuracy of our GNN models on the different datasets.

APPENDIX B
TRADITIONAL METRICS

In addition to the baselines mentioned in the paper, we also evaluate two heuristic metrics which are commonly used in link prediction tasks - the common neighborhood (CN) score and the Adamic-Adar (AA) score. The CN score will calculate the common neighbors between two nodes, while the AA score will calculate the inverse logarithmic degree centrality of the common neighbors. We show the detection performance of the metrics on Cora and Citeseer in Table VII. We can observe from the tables that our approach can still outperform these baselines in detecting malicious edges.

APPENDIX C
PERFORMANCE OF DIFFERENT DETECTION PRIMITIVES ON REAL-WORLD DATASETS

In this section we will explore in-depth on the detection performance against different types of attacks based on our proposed detection primitives.

Single-edge attack. The result for Single-edge attack is shown in Table VIII. As we can see, the link prediction-based approaches perform quite well. The best approach is LinkPred + GraphGenDetect, achieving an average of over 0.9 AUC over the tasks. When comparing the performance of the combination model LinkPred + GraphGenDetect with GGD, we see that the combination model significantly improves the detection performance as the LinkPred filters out the malicious edges (i.e., AUC $>$ 0.5). This shows that our GraphGenDetect detection model can learn "benign properties" from normal graphs and improve detection performance. On the other hand, a direct GraphGenDetect can achieve good performance in some cases. This shows that GraphGenDetect is a good yet unstable model, so a filtering algorithm can be combined to reduce the variation and improve its robustness.

In addition, We observe that the performance of LP + GGD in Cora is slightly better than in Citeseer while Rule has the best performance. This is because that the graph of Citeseer is more sparse than Cora (with more nodes and fewer edges), and Rule is the most densne graph. After filtering, the information contained in Citeseer dataset is reduced and therefore it is harder for the model to learn useful patterns.

Multi-edge direct attack. The result for Multi-edge direct attack is shown in Table VIII. We see that as the node degree increases, approaches related to link prediction algorithms perform no better than random guessing. As we have discussed in Section IV-B, we attribute this to a principle of the 'collective power of malicious edges'. By contrast, this collective power does not fool the OutlierDetect approach; its average AUC is over 0.85 when target node degree is larger than five, and 0.9 when it is larger than ten.

Multi-edge indirect attack. The result for Multi-edge indirect attack is shown in Table VIII. We see that the detection AUC decreases compared with Single-edge attack. This is reasonable: since more malicious edges are added, the defense would be more challenging. We observe that the LinkPred approach is affected the most. Therefore, the filtering algorithm does not help to improve the performance of GraphGenDetect. The best approach here is GraphGenDetect. One surprising observation is that the performance in Citeseer is better than Cora. We think that this phenomenon is also because of the sparsity: malicious edges tend to accumulate in a small neighbourhood of the target node, and the sparsity induces that subgraphs of nodes far away from the target node will not contain malicious edges. Therefore, the proportion of benign subgraphs will increase and therefore the trained generative detection model can learn a better pattern of benign properties.

Meta attack. The result for Meta attack is shown in Table VIII. We see that none of the approaches shows a very good performance against such kind of attack, especially over the Citeseer dataset. This is because this meta attack may be a combination of different type of attacks, and a large number of malicious edges are added (5%). Nevertheless, we can still see that our EDoG pipeline reaches acceptable performance on over the tasks. The detection performance on the Rule dataset is still very good, since it is a dense graph and contains more information for detection.

APPENDIX D
RESULTS ON DETECTING RANDOM EDGES ON GRAPHS

In this experiment, we randomly choose 1, 2, 4, 8, and 16 unconnected node pairs in Cora and Citeseer datasets to add random edges. We then run our detection pipeline as well as the baseline approaches on the graph to see whether these random edges will be identified as the malicious edges. In Table IX, we show the ratio of detecting non-random edges. Hence, larger value means that the detection model will not be distracted by the added random edges. In particular, 50% means that the detection method will not distinguish random and benign edges which is a desired property.

We can observe that baseline approaches tend to detect random edges as malicious edges while EDoG will ignore most random added edges. This is because these baseline approaches are essentially designed to detect abnormality in the graphs and therefore they will recognize the abnormal behaviour of the randomly added edges. On the other hand, our detection pipeline EDoG will not identify the random edges as malicious ones. In some cases the value is near 50% which means that the model views random edge to be

TABLE VII: Detection performance comparison between EDoG and traditional heuristic metrics against the malicious edges.

Task	Method	Single-edge attack	Multi-edge direct attack	Multi-edge indirect attack	Meta attack
Cora	CN	76.94	51.99	74.89	64.69
	AA	76.94	48.55	74.61	64.59
	EDoG	**90.99**	**84.91**	**83.91**	**68.84**
Citeseer	CN	31.29	46.05	65.40	51.43
	AA	27.71	40.78	64.07	49.47
	EDoG	85.50	81.17	72.69	52.23

TABLE VIII: The AUC of the detection components we proposed against different types of attacks. Here we explicitly consider target node with various degrees to demonstrate the generalization of our detection methods. Different components have their own advantages under certain setting.

		Single-edge attack		Multi-edges direct attack				Multi-edges indirect attack		Meta attack
	Node degree	$[1, 5]$	$(5, +\inf)$	$[1, 5]$	$(5, 10]$	$(10, 15]$	$(15, +\infty)$	$[1, 10]$	$(10, +\inf)$	-
Cora	LP	0.8960	0.8944	0.8319	0.6564	0.3787	0.3224	0.8506	0.8136	0.7935
	OD	0.2946	0.5233	0.4326	0.7378	0.9144	0.9123	0.2122	0.3588	0.4379
	GGD	0.6876	0.8535	0.6156	0.6864	0.4712	0.4643	0.6907	0.8156	0.5749
	LP + GGD	0.9088	0.9110	0.8319	0.6564	0.3787	0.3224	0.8368	0.8414	0.7568
	EDoG	0.9478	0.8794	0.7618	0.6910	0.6368	0.6202	0.6985	0.8080	0.6884
Citeseer	LP	0.8612	0.7612	0.5954	0.1633	0.2155	0.3181	0.6582	0.5924	0.3221
	OD	0.0858	0.5211	0.3457	0.8024	0.9176	0.9220	0.4498	0.3874	0.5939
	GGD	0.5547	0.5972	0.8137	0.6945	0.6834	0.5868	0.9068	0.7530	0.5232
	LP + GGD	0.8350	0.8750	0.6047	0.2505	0.4027	0.3910	0.7827	0.6712	0.3681
	EDoG	0.7051	0.6894	0.7219	0.5524	0.7217	0.6248	0.9018	0.6574	0.5223
Rule	LP	0.9845	0.9885	0.9370	0.9787	0.9631	0.9675	0.9932	0.9021	0.8658
	OD	0.9672	0.9836	0.9946	0.9865	0.9841	0.9832	0.9934	0.9503	0.9579
	GGD	0.7869	0.8833	0.7979	0.7025	0.8175	0.7355	0.9032	0.8207	0.7880
	LP + GGD	0.9481	0.9382	0.9331	0.8713	0.9015	0.8320	0.9119	0.9297	0.9315
	EDoG	0.9679	0.9767	0.9633	0.9297	0.9410	0.8971	0.9658	0.9413	0.9724

TABLE IX: The ratio of detecting non-random edges on Cora and Citeseer dataset. Larger value indicates that the model does not detect the added random edges, and 50% means the detection method will not distinguish random and benign edges, which is desired (Results that outperform the baselines by more than 3% are highlighted).

	Cora					Citeseer				
#Random edge	1	2	4	8	16	1	2	4	8	16
ALD	10.98%	17.54%	26.15%	11.88%	15.00%	4.35%	27.42%	11.37%	**30.51%**	29.11%
Katz	2.16%	7.74%	19.83%	7.18%	5.47%	7.07%	4.16%	7.11%	13.61%	8.56%
EDoG	9.64%	**52.35%**	**34.52%**	**30.74%**	17.43%	**36.32%**	**42.21%**	**23.32%**	22.21%	**45.19%**

similar as normal ones. This is ideal for our goal since our model will not be distracted by the random edges in the task of malicious edge detection. Comparing the result with the baseline approaches, we claim that our EDoG pipeline can not only detect malicious behaviors of the attacker, but also resilient against random added edges, which has great potential to improve the robustness of GNNs.