Rethinking the Trigger-injecting Position in Graph Backdoor Attack

1st Jing Xu
Delft University of Technology
Delft, Netherlands
j.xu-8@tudelft.nl

2nd Gorka Abad
Radboud University
Nijmegen, Netherlands
Ikerlan research centre
Arrasate-Mondragón, Spain
abad.gorka@ru.nl

3rd Stjepan Picek
Radboud Univesity
Nijmegen, Netherlands
Delft University of Technology
Delft, Netherlands
stjepan.picek@ru.nl

***Abstract*—Backdoor attacks have been demonstrated as a security threat for machine learning models. Traditional backdoor attacks intend to inject backdoor functionality into the model such that the backdoored model will perform abnormally on inputs with predefined backdoor triggers and still retain state-of-the-art performance on the clean inputs. While there are already some works on backdoor attacks on Graph Neural Networks (GNNs), the backdoor trigger in the graph domain is mostly injected into random positions of the sample. There is no work analyzing and explaining the backdoor attack performance when injecting triggers into the most important or least important area in the sample, which we refer to as trigger-injecting strategies MIAS and LIAS, respectively. Our results show that, generally, LIAS performs better, and the differences between the LIAS and MIAS performance can be significant. Furthermore, we explain these two strategies' similar (better) attack performance through explanation techniques, which results in a further understanding of backdoor attacks in GNNs.**

***Index Terms*—backdoor attack, trigger-injecting position, graph neural networks**

I. Introduction

Graph Neural Networks (GNNs) have demonstrated their superior performance in a variety of applications, such as node classification [6], graph classification [2], image classification [30], and natural language processing [30]. However, GNNs are vulnerable to various adversarial attacks, including the backdoor attack. Specifically, a backdoor attack occurs when the adversary deliberately modifies a proportion of the training data by adding the trigger (e.g., subgraph in a graph) to make the model misclassify the samples with the trigger as the target label(s). The backdoored GNN model aims to perform normally on benign testing samples. However, if the same trigger used in the training phase is introduced onto a testing sample, the backdoored model exhibits a particular output behavior of the adversary's choosing (e.g., misclassification into to target label(s)). Backdoor attacks have been demonstrated to perform malicious tasks on security-related graph learning services, such as converting the label of a fraud account to benign in a social network [10]. Hence, the backdoor attack is a serious threat to the practical applications of GNNs.

Several works explored the backdoor attacks in GNNs [22], [24], [28]. In these works, one idea of the trigger-injecting position is randomly selecting a subgraph as there is no specific location information in a graph [21]. Another idea to inject the trigger into a graph is to select the subgraph which has high similarity with the trigger graph [22]. Moreover, based on the improvement of the explanation techniques in the graph domain, [24] proposed injecting the trigger into the most important or least important area of the sample. However, that work does not provide any experimental analysis to confirm the assumptions made. Also, there is no work so far on using explanation tools to explain the backdoor attack behavior in the graph domain. This work first raises a core question:

What is the attack performance when injecting trigger into the most or least important area of the sample?

To answer this question, we explore the impacts of the backdoor trigger-injecting position from the perspective of the most (MIAS) or least important area of the sample (LIAS). Although there is no location information in a graph, we can still locate the most (least) important area in a graph, like in an image, by using some explanation techniques [25]. As shown in experiments, we demonstrate that the attack performance of LIAS is better, where the difference from MIAS can even be significant. This observation inspires one further question:

Can we explain this difference?

There are already some works on explaining backdoor attacks in the image domain through visualization techniques [3], [23]. For example, [3] plotted the average activations of the backdoored model's last convolutional layer over clean and backdoored images to explain their attack. [23] used the Grad-CAM [15] visualization method to explain the backdoor attack in federated learning. One example of explaining a backdoor attack in the image domain with Grad-CAM is shown in Fig. 1. Comparing the heatmaps of the clean and poisoned images on the backdoored model, we can clearly understand how the backdoored model recognizes the trigger pattern to achieve the backdoor attack. In contrast, applying visualization techniques to explain the backdoor attack behavior in the graph domain is difficult. First, the complexity of the visual representation of a graph is much larger than visualizing an image, especially for large graphs [5]. Second, visualizing the graph neural networks to explain the backdoor attack is not trivial as it is a time-consuming or even impossible process [7].

Therefore, in this work, instead of using the visualization method, we explain the difference between two trigger-injecting strategies by computing an evaluation metric. Specifically, we compute the similarity of the predicted mask of the representative features from the backdoored model and the target mask of the representative features from the clean model. In our experiments, we find that the successfully misclassified samples generally have high similarity while the unsuccessfully misclassified samples have a much lower similarity. However, we also find that in one specific case, the high similarity does not lead to a successful attack. We further study this phenomenon and find that the backdoored model trained by MIAS can recognize the original feature pattern in addition to the trigger pattern.

Our work is the first to revisit the trigger-injecting position in graph backdoor attacks and provide a new perspective. Our key contributions are:

1) We investigate backdoor attacks in GNNs by injecting triggers into the most or least important area of the sample.
2) We design a novel explanation framework to analyze the causes of the difference between these two strategies.
3) We verify the difference with quantitative analysis (recall score), which helps us further understand the backdoor attack behavior in GNNs.

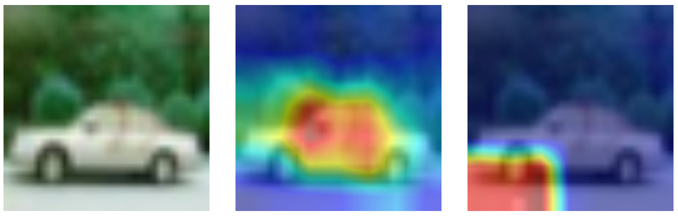


Fig. 1: An example of using Grad-CAM to explain a backdoor attack in the image domain. (a) clean image, (b) heatmap of clean image for the true label on the backdoored model (predicted as the true label), (c) heatmap of the poisoned image for the target label on the backdoored model (predicted as the target label).

II. Background

A. Graph Neural Networks

Recently, Graph Neural Networks (GNNs) have achieved significant success in processing non-Euclidean spatial data, which are common in many real-world scenarios [30]. Unlike traditional neural networks, e.g., Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs), GNNs work on graph data. GNNs take a graph $G = (V, E, X)$ as an input, where V, E, X denote nodes, edges, and node attributes, and learn a representation vector (embedding) for each node $\boldsymbol{v} \in G$, $z_{\boldsymbol{v}}$, or the entire graph, z_G.

In particular, in modern GNNs, the node representation is computed by recursive aggregation and transformation of feature representations of its neighbors. After k iterations of aggregation, a node's representation captures both structure and feature information within its k-hop network neighborhood. Formally, the k-th layer of a GNN is:

$$x_{\boldsymbol{v}}^{(k)} = AGGREGATION^{(k)}(\left\{z_{\boldsymbol{v}}^{(k-1)}, \left\{z_{\boldsymbol{u}}^{(k-1)} | \boldsymbol{u} \in \mathcal{N}_{\boldsymbol{v}}\right\}\right\}), \tag{1}$$

$$z_{\boldsymbol{v}}^{(k)} = TRANSFORMATION^{(k)}(x_{\boldsymbol{v}}^{(k)}), \tag{2}$$

where $z_{\boldsymbol{v}}^{(k)}$ is the representation of node $\boldsymbol{v}$ computed in the k-th iteration. $\mathcal{N}_{\boldsymbol{v}}$ are 1-hop neighbors of node $\boldsymbol{v}$, and the $AGGREGATION(\cdot)$ is an aggregation function that can vary for different GNN models. $z_{\boldsymbol{v}}^{(0)}$ is initialized as node feature. The $TRANSFORMATION(\cdot)$ function consists of a learnable weight matrix and activation function. For the node classification task, the node representation $z_{\boldsymbol{v}}$ is used for prediction. In this paper, we investigate the node classification task. Moreover, we focus on two representation models of this family, which differ in one of the above two steps: aggregation and transformation. In the following, we briefly describe these models and their differences.

Graph Convolutional Networks (GCN) [8]. Let $d_{\boldsymbol{v}}$ denotes the degree of node $\boldsymbol{v}$. The aggregation operation in GCN is then given as:

$$x_{\boldsymbol{v}}^{(k)} \leftarrow \sum_{u \in \mathcal{N}_{\boldsymbol{v}} \bigcup \boldsymbol{v}} \frac{1}{\sqrt{d_{\boldsymbol{v}} d_{\boldsymbol{u}}}} z_{\boldsymbol{u}}^{(k-1)}.$$

GCN performs a non-linear transformation over the aggregated features to compute the node representation at layer k:

$$z_{\boldsymbol{v}}^{(k)} \leftarrow ReLU(x_{\boldsymbol{v}}^{(k)} W^{(k)}).$$

Graph Attention Networks (GAT) [19]. In addition to the standard neighbor aggregation scheme mentioned above in equation 1 and equation 2, there are other non-standard neighbor aggregation schemes, e.g., weighted average via attention in GAT. Specifically, given a shared attention mechanism a, attention coefficients can be computed by:

$$e_{\boldsymbol{vu}} = a(W z_{\boldsymbol{v}}^{(k-1)}, W z_{\boldsymbol{u}}^{(k-1)}) \tag{3}$$

that indicate the importance of node $\boldsymbol{u}$'s features to node $\boldsymbol{v}$. Then, the normalized coefficients can be computed by using the softmax function:

$$\alpha_{\boldsymbol{vu}} = softmax_{\boldsymbol{u}}(e_{\boldsymbol{vu}}). \tag{4}$$

Finally, the next-level feature representation of node $\boldsymbol{v}$ is:

$$z_{\boldsymbol{v}}^{(k)} = \sigma\left(\frac{1}{P}\sum_{p=1}^{P}\sum_{\boldsymbol{u}\in\mathcal{N}_{\boldsymbol{v}}} \alpha_{\boldsymbol{vu}}^{p} W^{p} z_{\boldsymbol{u}}^{(k-1)}\right), \tag{5}$$

where $\alpha_{\boldsymbol{vu}}^{p}$ are the normalized coefficients computed by the p-th attention mechanism a^p and W^p is the corresponding input linear transformation's weight matrix.

B. Backdoor Attacks

Backdoors are training time attacks that aim to achieve misclassification at the testing phase for trigger-embedded samples while working correctly on clean inputs. Several studies showed that GNNs are also vulnerable to backdoor attacks. Similar to the backdoor attack in CNNs, the backdoor attack in GNNs can be implemented by poisoning the training data with a trigger, which can be a subgraph with/without features [22], [28] or a subset of node features [24]. After training the GNN model with the trigger-embedded data, the backdoored GNN would predict the test example injected with a trigger as the pre-defined target label.

C. Explainability of GNNs

Recently, several explainability techniques in GNNs have been proposed, such as XGNN [26], GNNExplainer [25], PGExplainer [11], and SubgraphX [27]. These methods are developed from different angles and provide different levels of explanations.

GNNExplainer is the model-agnostic approach for providing explanations on any GNN-based model's predictions. Given a trained GNN model and its prediction(s), GNNExplainer returns an explanation in the form of a small subgraph of the input graph, with a small subset of node features that contribute most to the final model prediction(s). In this paper, we focus on the GNNExplainer method as it can explain predictions of any GNN on any graph-based machine learning task without requiring modification of the underlying GNN architecture or re-training.

III. Threat Model

We consider a *gray-box* threat model assuming the attacker can freely modify a small portion of the training dataset and has no knowledge about the training algorithms or the models used by the victims. We also assume the attacker performs a *dirty-label* backdoor attack, where the poisoned samples' labels are changed to the target label. Although this kind of attack is weaker than *clean-label* backdoor attacks [18], where the labels remain unaltered, dirty label attacks are the most common in the literature [22], [24], [28]. The attacker's goal is to inject a backdoor in the given pre-trained clean GNN model through training over the poisoned training dataset, which achieves misclassification under the presence of a trigger while maintaining clean high accuracy on the original task. This threat model is realistic in real-world settings. For example, if the training dataset is collected from public users, the adversary can provide trigger-embedded training data to implement the backdoor attack.

IV. Methodology

A. General Framework

As stated before, we aim to discover if and how the explainability techniques in GNNs help improve the performance of backdoor attacks. Here, we focus on utilizing the feature-trigger backdoor attack from [24] for the node classification task. Generally, two steps are conducted:

(1) We apply an explainability technique (i.e., GNNExplainer) on a pre-trained clean GNN model to implement backdoor attacks based on two trigger-injecting strategies defined below.

Definition 1 (The Most/Least Representative Features): Through applying the GNNExplainer on the pre-trained clean GNN model on the target node, we can obtain the original importance order of the node features. Based on the importance order information, we can locate the most or least representative features.

Definition 2 (Most Important Area Strategy (MIAS)): We select the most representative features of the target node and inject the feature trigger into the corresponding dimensions.

Definition 3 (Least Important Area Strategy (LIAS)): We select the least representative features of the target node and inject the feature trigger into the corresponding dimensions.

We then compare the attack performance based on these two strategies, including the attack success rate and clean accuracy drop.

(2) Next, we try to explain the attack performance of these two strategies by again applying the explainability techniques on the backdoored model over the poisoned testing dataset. As a result, we can obtain the new importance order of the node features, which is used to compute the similarity with the original feature importance order. The proposed framework is presented in Fig. 2.

B. Explanation Design

The detailed process of generating poisoned training dataset and target masks is presented in Algorithm 1. $EXP(\cdot)$ is the applied GNN explanation technique, i.e., GNNExplainer, and s is the trigger-injecting strategies, i.e., MIAS or LIAS. The algorithm first samples a subset from the original training dataset with a poisoning rate r (line 2). For each sampled node, the algorithm will compute the corresponding feature order to determine the trigger-injecting location for MIAS and LIAS. Meanwhile, the label of the poisoned training dataset will change to the target label. The trigger size n is the number of the features in the feature trigger, which means n node features will be modified. The poisoned testing dataset is obtained by injecting a trigger (following the same strategy as the poisoned training dataset) into the samples and changing their labels to the target label. Finally, based on the order of representative features, we can generate a target mask for each node in the poisoned testing dataset (line 17). The target mask has the same shape as the node feature vector, and the most (least) n important features are masked in while other features are masked out. To evaluate whether the backdoored model can recognize the trigger pattern precisely, the number of features to be masked in is set to be n. The target mask indicates for the target node which features contribute more or less to the final prediction from the **pre-trained clean model** θ.

Once the poisoned training dataset is generated, we can obtain the backdoored models $\hat{\theta}^s$ by retraining the clean model

Algorithm 1: Generate Poisoned Training Dataset and Target Masks

```
Input: Pre-trained clean GNN model $\theta$, Training set
       $D_{train}$, Testing set $D_{test}$, Trigger-injecting strategy
       $s \in \{MIAS, LIAS\}$, Target label $y_t \in [0, C)$
Output: Poisoned training dataset $\hat{D}^s_{train}$, Poisoned testing
        dataset $\hat{D}^s_{test}$, Target masks $M^s_t$
1  /* Sampling Training Dataset to Inject Trigger */
2  $\hat{D}^s_{train} \leftarrow sample(D_{train}, r, y \neq y_t)$
3  foreach $\{x, y\} \in \hat{D}^s_{train}$ do
4      /* Computing Order of Representative Features */
5      $feature\_order = EXP(\theta, x, y)$
6      $\hat{x}^s = Inject\_Trigger(x, feature\_order, s)$
7      $\hat{y}^s = y_t$
8  end
9  $\hat{D}^s_{test} \leftarrow D_{test}[\backslash y_t]$
10 $M^s_t \leftarrow \emptyset$
11 foreach $\{x, y\} \in \hat{D}^s_{test}$ do
12     /* Computing Order of Representative Features */
13     $feature\_order = EXP(\theta, x, y)$
14     $\hat{x}^s = Inject\_Trigger(x, feature\_order, s)$
15     $\hat{y}^s = y_t$
16     /* Generating Target Mask */
17     $m^s_i = Get\_Mask(feature\_order, s)$
18     $M^s_t = M^s_t \cup m^s_i$
19 end
20 return $\hat{D}^s_{train}, \hat{D}^s_{test}, M^s_t$
```

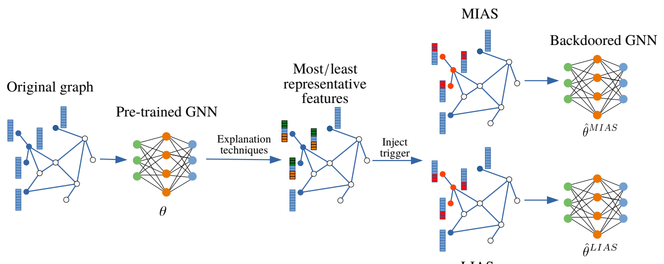
Algorithm 2: Train Backdoored GNN Models and Generate Predicted Masks

```
Input: Pre-trained clean GNN model $\theta$, Training set
       $D_{train}$, Poisoned training dataset $\hat{D}_{train}$, Poisoned
       testing dataset $\hat{D}_{test}$, Trigger-injecting strategy
       $s \in \{MIAS, LIAS\}$
Output: Backdoored GNN model $\hat{\theta}^s$, Predicted Masks $M^s_p$
1  /* Training Backdoored Models */
2  /* $\{x, y\} \in D_{train}$, $\{\hat{x}^s, \hat{y}^s\} \in \hat{D}^s_{train}$ */
3  $\hat{\theta}^s = \text{argmin}_\theta(\sum_i L(x_i, y_i; \theta) + \sum_i L(\hat{x}^s_i, \hat{y}^s_i; \theta))$
4  $M^s_p \leftarrow \emptyset$
5  foreach $\{\hat{x}^s, \hat{y}^s\} \in \hat{D}^s_{test}$ do
6      /* Getting Predictive Mask */
7      $feature\_order = EXP(\hat{\theta}^s, \hat{x}^s, \hat{y}^s)$
8      $m^s_i = Get\_Mask(feature\_order)$
9      $M^s_p = M^s_p \cup m^s_i$
10 end
11 return $\hat{\theta}^s, M^s_p$
```

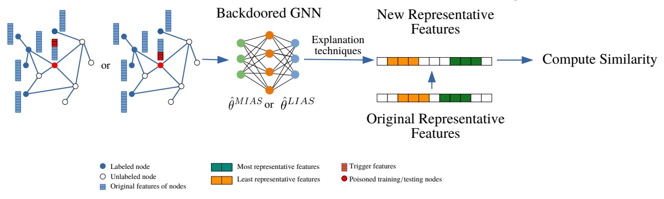
θ with the backdoored training dataset. [1] The process of training the backdoored models and obtaining predicted masks is shown in Algorithm 2. To analyze the impact of injecting trigger into the most/least important part of the node features on the attack performance, we compare the attack performance of $\hat{\theta}^{MIAS}$ and $\hat{\theta}^{LIAS}$, including the attack success rate and clean accuracy drop. Finally, for the poisoned testing dataset, which we used to calculate the attack success rate, we again utilize the GNNExplainer to obtain the new feature importance order for each node on the the backdoored GNN model $\hat{\theta}^{MIAS}$ or $\hat{\theta}^{LIAS}$ (line 7). The new feature importance order is used to generate the predicted mask which shows the importance of each feature for the final prediction from the **backdoored GNN model**. Combining the target masks we get in Algorithm 1, we can compute the similarity between the ordering of the new representative features and the old ones by calculating the recall score of the target mask and the predicted mask:

$$RS^s_i = \frac{TP(M^s_{t,i}, M^s_{p,i})}{TP(M^s_{t,i}, M^s_{p,i}) + FN(M^s_{t,i}, M^s_{p,i})}, i \in N$$
$$M^s_{t,i}(M^s_{p,i}) = [0, \cdots, 1, \cdots, 1, \cdots, 0], \tag{6}$$

where RS^s_i is the recall score of the ith poisoned testing sample with s strategy, $M^s_{t,i}$ and $M^s_{p,i}$ is the target mask, and predictive mask of the ith poisoned testing sample, TP and FN is the true positive and false negative rate of these two masks, respectively, and N is the number of the poisoned testing dataset. We assume that higher similarity indicates that the backdoored model can better recognize the trigger pattern, contributing to better attack performance.



(a) Backdoor attacks based on two strategies.


or
Backdoored GNN
Explanation techniques
New Representative Features
Compute Similarity
$\hat{\theta}^{MIAS}$ or $\hat{\theta}^{LIAS}$
Original Representative Features
Labeled node
Unlabeled node
Original features of nodes
Most representative features
Least representative features
Trigger features
Poisoned training/testing nodes

(b) Procedure of explaining two attack strategies.

Fig. 2: An illustration of backdoor attack and explanation framework.

V. Experimental Results

A. Experimental Setting

We implemented the backdoor attack on the node classification task using the PyTorch framework. All experiments were run on a server with 2 Intel Xeon CPUs, 1 NVIDIA 1080 Ti GPU with 32GB RAM, and Ubuntu 20.04 LTS OS. Each experiment was repeated 10 times to obtain the average result.

Dataset. For our experiments, we use two publicly available real-world datasets for the node classification task: Cora [16] and CiteSeer [16]. These two datasets are citation networks in which each publication is described by a binary-valued word vector indicating the absence/presence of the corresponding

[1]In this work, we combine the original training dataset and the poisoned training dataset as the backdoored training dataset.

word in the collection of $1,433$ and $3,703$ unique words, respectively.

For each node classification dataset, we split 20% of the total nodes as the original training dataset (labeled), and the rest of the nodes are treated as the original testing dataset. To generate the backdoored training dataset, we sample 10% of the original training dataset to inject the feature trigger and relabel these nodes with the target label. The trigger size is set to 5% of the total number of node feature dimensions. We set these parameters as they provided the best results after conducting a tuning phase.

Models and training. We use the popular GAT [19] and GCN [8] models, as these two methods are state-of-the-art GNN models for the node classification task. We train the clean and backdoored GNN models with a learning rate of 0.005 and use Adam as the optimizer.

Attack evaluation metrics. To compare the attack performance of MIAS and LIAS, we utilize two commonly used backdoor attack evaluation metrics:

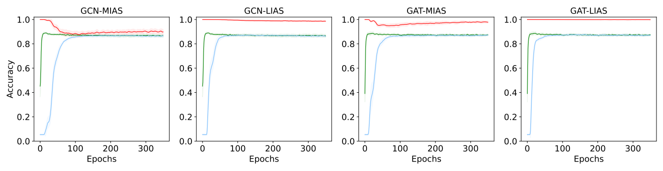
1) **Attacks Success Rate** (ASR): measures the backdoor performance of the model on a fully poisoned dataset $\hat{D}$. It is computed as $ASR = \frac{\sum_{i=1}^{N} \mathbb{I}(\hat{\theta}(\hat{x_i})=y_t)}{N}$ where $\hat{\theta}$ is the poisoned model, $\hat{x_i}$ is a poisoned input, $\hat{x_i} \in \hat{D}$, y_t is the target class, and $\mathbb{I}$ is an indicator function.
2) **Clean Accuracy Drop** (CAD): measures the effect of the backdoor attack on the original task. It is calculated by comparing the performance of the poisoned and clean models on a clean holdout testing set. The accuracy drop should generally be small to keep the attack stealthy.

B. Results and Analysis

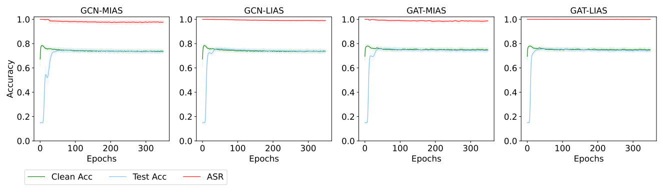
Results. The backdoor attack results on two graph datasets based on two models and two trigger-injecting strategies are shown in Fig. 3. In particular, the ASR and CAD of two GNN models on two datasets are presented in Table I. We can observe that both strategies can achieve a high attack success rate, i.e., more than 97%, except GCN on the Cora dataset with MIAS. In addition, in most cases, the ASR of LIAS is slightly higher, around 1%, than that of MIAS. However, for the GCN model on the Cora dataset, the ASR of LIAS is significantly higher: more than 8%, than the MIAS. We can also see that the CAD for all datasets and models is unnoticeable, and the difference between the two strategies over CAD is negligible.

Analysis. Next, we investigate the reason why the backdoor attack performance of the LIAS is somewhat higher or significantly higher (for the GCN model on the Cora dataset) than the MIAS. As mentioned in Section IV, we calculate the recall score of the target mask and the predicted mask to evaluate the similarity between the ordering of the new representative features and the old ones.

The histogram of recall scores over the poisoned testing dataset of all datasets and models is shown in Fig. 4. We can observe that most poisoned testing samples have a recall score of more than 0.5 in both MIAS and LIAS, which results in a high attack success rate for both strategies. To further investigate the slight advantage of the LIAS over the MIAS, we split the poisoned testing samples into two parts, one is misclassified into the target class successfully, and the other one is not, and compute the recall scores for these two parts, as shown in Fig. 5. We notice that, generally, the successfully misclassified nodes have significantly higher recall scores than those not misclassified into the target class. This phenomenon is consistent with the assumption mentioned in Section IV, i.e., the higher similarity between the ordering of the new representative features and that of the original ones indicates that the backdoored model can recognize the trigger pattern better. When comparing the second column and the last column of Fig. 5b, 5c, and 5d, we also see that LIAS has fewer nodes with low recall score than MIAS, which we believe is the reason of higher ASR of LIAS than MIAS.



(a) Cora.



(b) CiteSeer.

Fig. 3: Backdoor attack results of two trigger-injecting strategies.

In contrast, we surprisingly see that for the GCN model on the Cora dataset with MIAS, the unsuccessfully misclassified nodes also have a high recall score as the successfully misclassified nodes. We assume that the main reason behind this is that, under the MIAS, the feature trigger is injected into the positions of the most representative features. Thus the backdoored model will recognize not only the trigger pattern but also the representative feature pattern for the original label. Therefore, for MIAS, it is possible that even the poisoned testing samples that are not successfully misclassified into the target class will have a high recall score. We verify this hypothesis by extending the target masks and predicted masks twice the feature trigger length, i.e., $2 * n$, and computing the recall scores again. [2] The histogram of the new recall scores of the GCN model on the Cora dataset is shown in Fig. 6. We also checked the prediction of the backdoored model over the unsuccessfully misclassified nodes. The output indicates that all these nodes are classified into their original classes. Comparing Fig. 5a and 6, we observe that the recall scores of the successfully misclassified nodes generally reduce

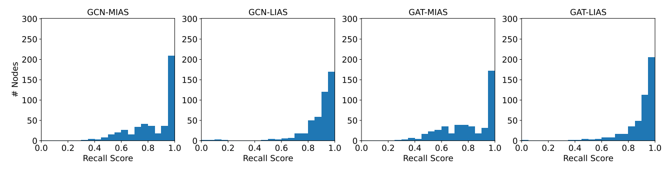
[2]Here, we select an extension rate of 2. To verify the hypothesis, the extension rate can be set to $\gamma > 1$, and the recall scores of the successfully misclassified nodes are expected to reduce to $1/\gamma$ of that without extended masks.

TABLE I: Backdoor attack performance of MIAS and LIAS (SD: standard deviation).

MIAS				
Dataset	GCN		GAT	
	ASR ± SD	CAD ± SD	ASR ± SD	CAD ± SD
Cora	90.08% ± 0.29%	0.32% ± 0.19%	97.91% ± 0.12%	0.34% ± 0.24%
CiteSeer	97.70% ± 0.10%	0.32% ± 0.17%	98.54% ± 0.09%	0.71% ± 0.20%

LIAS				
Dataset	GCN		GAT	
	ASR ± SD	CAD ± SD	ASR ± SD	CAD ± SD
Cora	98.65% ± 0.06%	0.27% ± 0.21%	99.89% ± 0.03%	0.27% ± 0.21%
CiteSeer	98.96% ± 0.07%	0.15% ± 0.18%	99.88% ± 0.03%	0.80% ± 0.17%

to half of that without extended masks. We believe this is because, for these nodes, the backdoored model recognizes the trigger location exactly, and when we extended the masks twice the trigger length, only half of the features can be recalled. However, we can also see that for the MIAS, the recall scores of the unsuccessfully classified nodes are still as high as those without the extended masks. This is because the backdoored model recognizes the feature pattern for the original label (that is why these nodes are classified into the original class and the attack is not successful), so even if the masks are extended, the recall score is still high.


(a) Cora.

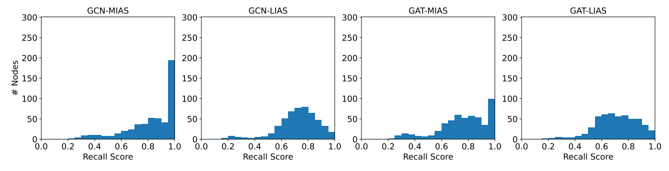
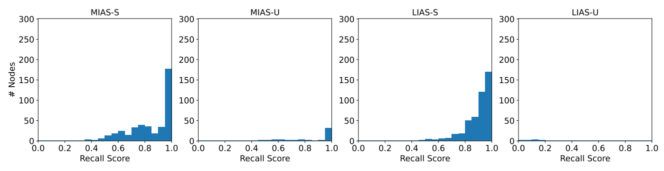
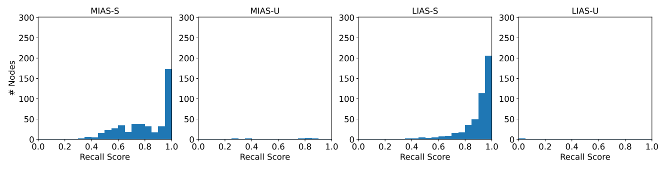

(b) CiteSeer.

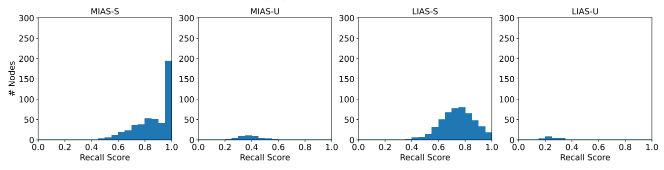
Fig. 4: Histogram of recall scores over the poisoned testing dataset.

VI. Related Work

Backdoor Attacks in GNNs Several works have conducted backdoor attacks in GNNs. Zhang et al. presented a subgraph-based backdoor attack in GNNs for graph classification task [28]. Xi et al. proposed a subgraph-based backdoor attack in GNNs, for both node classification and graph classification tasks [22]. Xu et al. explored the trigger-injecting position for the graph backdoor attack [24], representing the most related work to our paper. However, in that paper, the authors only provided assumptions about the results, and no experimental analysis was given to confirm the assumptions. In this work, we give an empirical analysis of the attack results, which leads


(a) Cora-GCN.


(b) Cora-GAT.


(c) CiteSeer-GCN.

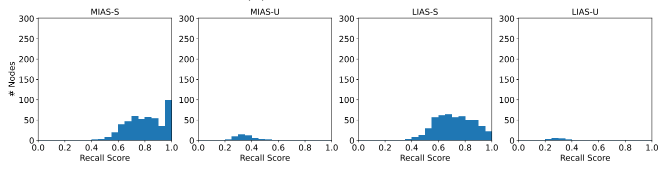

(d) CiteSeer-GAT.

Fig. 5: Histogram of recall scores over two parts of the poisoned testing dataset (S means the nodes are successfully misclassified into the target label, U means not).

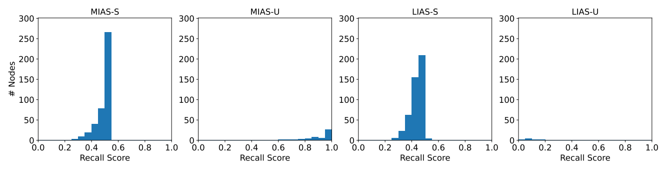


Fig. 6: Histogram of recall scores of GCN model on Cora dataset with extended masks (S means the nodes are successfully misclassified into the target label, U means not).

to further understanding of the backdoor attack behavior in GNNs.

Explanability in GNNs GNNs have become increasingly popular since many real-world data can be naturally represented as graphs, such as social networks, chemical molecules, and financial data [4], [29]. Consequently, numerous approaches are proposed to explain the predictions of GNNs. Generally, these methods can be categorized into two mainstream lines of research. One is the parametric explanation methods that are widely used nowadays. For instance, GNNExplainer [25] learns soft masks for edges and node features to explain the predictions via mask optimization. The soft masks are randomly initialized and treated as trainable variables. [11] proposed PGExplainer to collectively explains multiple instances with a probabilistic graph generative model. XGNN [26] uses a graph generator to generate class-wise graph patterns to explain GNNs for each class. Vu et al. proposed PGM-Explainer, a Bayesian network on the pairs of graph perturbations and prediction changes [20].

The other line is the non-parametric explanation methods, which do not involve any additional trainable models. They employ heuristics like gradient-like scores obtained by backpropagation as the feature contributions of a specific instance [1], [13], [14]

Explainability for Backdoor Attacks With the thriving development of explainability techniques in machine learning, the attacker can use model explanations to gain knowledge about the model to perform the adversarial attacks [12]. Kuppa et al. [9] used counterfactual explanations to find the malware features that most heavily impact the classifier decision. They used this knowledge to craft adversarial training samples that efficiently poison the model. Severi et al. [17] used SHAP to craft backdoor triggers in malware detectors. Utilizing the explanation, they determined which features to poison, resulting in a success rate of up to three times higher than that of a greedy algorithm that does not use explainable artificial intelligence (XAI). Xu et al. [24] injected backdoors into GNNs by leveraging XAI techniques. Although there is an increasing number of works on utilizing explanation techniques to implement backdoor attacks in deep learning models, there is no work on using explanation tools to explain the backdoor attack behavior in the graph domain.

VII. Conclusion and Future Work

This paper provides a comprehensive analysis and explanation of graph backdoor attacks with two trigger-injecting strategies; MIAS and LIAS. We investigate the node classification task and compare the attack performance for these two strategies. Our findings show that LIAS always achieves higher attack performance than MIAS. We further explain the difference with quantitative analysis, which contributes to a further understanding of the backdoor attack behavior in GNNs. Future work will include explaining the backdoor attack behavior of two trigger-injecting strategies in the graph classification task. More precisely, we would compute the similarity between the new representative subgraph and the old one by calculating the recall score of the target mask and the predicted mask.

References

[1] Federico Baldassarre and Hossein Azizpour. Explainability techniques for graph convolutional networks. *arXiv preprint arXiv:1905.13686*, 2019.

[2] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. *arXiv preprint arXiv:1912.09893*, 2019.

[3] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, 7:47230–47244, 2019.

[4] William L Hamilton. Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning*, 14(3):1–159, 2020.

[5] Yifan Hu and Lei Shi. Visualizing large graphs. *Wiley Interdisciplinary Reviews: Computational Statistics*, 7(2):115–136, 2015.

[6] Mengda Huang, Yang Liu, Xiang Ao, Kuan Li, Jianfeng Chi, Jinghua Feng, Hao Yang, and Qing He. Auc-oriented graph neural network for fraud detection. In *Proceedings of the ACM Web Conference 2022*, pages 1311–1321, 2022.

[7] Zhihua Jin, Yong Wang, Qianwen Wang, Yao Ming, Tengfei Ma, and Huamin Qu. Gnnlens: A visual analytics approach for prediction error diagnosis of graph neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 2022.

[8] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[9] Aditya Kuppa and Nhien-An Le-Khac. Adversarial xai methods in cybersecurity. *IEEE transactions on information forensics and security*, 16:4924–4938, 2021.

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

[11] Dongsheng Luo, Wei Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, Haifeng Chen, and Xiang Zhang. Parameterized explainer for graph neural network. *Advances in neural information processing systems*, 2020.

[12] Azqa Nadeem, Daniël Vos, Clinton Cao, Luca Pajola, Simon Dieck, Robert Baumgartner, and Sicco Verwer. Sok: Explainable machine learning for computer security applications. *arXiv preprint arXiv:2208.10605*, 2022.

[13] Phillip E Pope, Soheil Kolouri, Mohammad Rostami, Charles E Martin, and Heiko Hoffmann. Explainability methods for graph convolutional neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10772–10781, 2019.

[14] Thomas Schnake, Oliver Eberle, Jonas Lederer, Shinichi Nakajima, Kristof T Schütt, Klaus-Robert Müller, and Grégoire Montavon. Higher-order explanations of graph neural networks via relevant walks. *IEEE transactions on pattern analysis and machine intelligence*, 44(11):7581–7596, 2021.

[15] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[16] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 2008.

[17] Giorgio Severi, Jim Meyer, Scott E Coull, and Alina Oprea. Explanation-guided backdoor poisoning attacks against malware classifiers. In *USENIX Security Symposium*, pages 1487–1504, 2021.

[18] Alexander Turner, Dimitris Tsipras, and Aleksander Madry. Clean-label backdoor attacks. 2018.

[19] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *ICLR*, 2018.

[20] Minh Vu and My T Thai. Pgm-explainer: Probabilistic graphical model explanations for graph neural networks. *Advances in neural information processing systems*, 33:12225–12235, 2020.

[21] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[22] Zhaohan Xi, Ren Pang, Shouling Ji, and Ting Wang. Graph backdoor. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1523–1540, 2021.

[23] Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. Dba: Distributed backdoor attacks against federated learning. In *International conference on learning representations*, 2020.

[24] Jing Xu, Minhui Xue, and Stjepan Picek. Explainability-based backdoor attacks against graph neural networks. In *Proceedings of the 3rd ACM Workshop on Wireless Security and Machine Learning*, 2021.

[25] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems*, 32, 2019.

[26] Hao Yuan, Jiliang Tang, Xia Hu, and Shuiwang Ji. Xgnn: Towards model-level explanations of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 430–438, 2020.

[27] Hao Yuan, Haiyang Yu, Jie Wang, Kang Li, and Shuiwang Ji. On explainability of graph neural networks via subgraph explorations. In *International Conference on Machine Learning*. PMLR, 2021.

[28] Zaixi Zhang, Jinyuan Jia, Binghui Wang, and Neil Zhenqiang Gong. Backdoor attacks to graph neural networks. In *Proceedings of the 26th ACM Symposium on Access Control Models and Technologies*, pages 15–26, 2021.

[29] Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 34(1):249–270, 2020.

[30] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.