BACKDOOR ATTACKS TO GRAPH NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

In this work, we propose the first backdoor attack to graph neural networks (GNN). Specifically, we propose a *subgraph based backdoor attack* to GNN for graph classification. In our backdoor attack, a GNN classifier predicts an attacker-chosen target label for a testing graph once a predefined subgraph is injected to the testing graph. Our empirical results on three real-world graph datasets show that our backdoor attacks are effective with a small impact on a GNN's prediction accuracy for clean testing graphs.

1 INTRODUCTION

Graphs have been widely used to model complex interactions between entities. For instance, in online social networks, a user and its friends can be modeled as a graph (called *ego network* in network science), where the user and its online friends are nodes, and an edge between two nodes indicates online friendship or interaction between them. Likewise, a Bitcoin transaction can be modeled as an ego network, where the nodes are the transaction and the transactions that have Bitcoin flow with it, and an edge between two transactions indicates the flow of Bitcoin from one transaction to the other. *Graph classification*, which takes a graph as an input and outputs a label for the graph, is a basic graph analytics tool and has many applications such as fraud detection (Wang et al., 2017a; Weber et al., 2019; Wang et al., 2019a), malware detection (Kong & Yan, 2013; Nikolopoulos & Polenakis, 2017; Hassen & Chan, 2017; Yan et al., 2019), and healthcare (Li et al., 2017; Altae-Tran et al., 2017; Chen et al., 2018). Graph neural network (GNN) based graph classification has attracted increasing attention due to its superior prediction accuracy. Given a graph, a GNN uses a neural network to analyze the complex graph structure and predict a label for the graph. For instance, to detect fake users in online social networks, a user is predicted to be fake if a GNN predicts the label "fake" for the user's ego network. To detect fraudulent transactions in Bitcoin, a transaction is fraudulent if a GNN predicts the label "fraudulent" for the transaction's ego network.

Since GNNs are used for security applications, an attacker is motivated to attack GNNs to evade detection. For instance, a fake user can attack GNNs such that it is misclassified as a genuine user. However, GNN based graph classifications in such adversarial settings are largely unexplored. Most existing studies (Zügner et al., 2018; Bojchevski & Günnemann, 2019a; Wang & Gong, 2019; Zügner & Günnemann, 2019b) on GNNs in adversarial settings focused on *node classification* instead of graph classification. Node classification aims to predict a label for each node in a graph, while graph classification aims to predict a label for the entire graph. One exception is that Dai et al. (2018) proposed adversarial examples to attack GNN based graph classification, where an attacker perturbs the structure of a testing graph such that the target GNN misclassifies the perturbed testing graph (i.e., the perturbed testing graph is an adversarial example). However, such attacks require optimized (different) perturbations for different testing graphs and have limited success rates when the target GNN is unknown (Dai et al., 2018).

Our work: In this work, we propose the first *backdoor attack* to GNNs. Unlike adversarial examples, a backdoor attack applies the same *trigger* to testing graphs and does not need knowledge of the target GNN to be successful. Backdoor attacks have been extensively studied in the image domain (Gu et al., 2017; Chen et al., 2017a; Liu et al., 2018b; Li et al., 2018; Tran et al., 2018; Yao et al., 2019; Salem et al., 2020). However, backdoor attacks to GNNs are unexplored. Unlike images whose pixels can be represented in a Cartesian coordinate system, graphs do not have such Cartesian coordinate system and graphs to an GNN can have different sizes.

We propose a *subgraph based backdoor attack* to GNN based graph classification. Specifically, we propose to use a subgraph pattern as a backdoor trigger, and we characterize our subgraph based backdoor attack using four parameters: *trigger size*, *trigger density*, *trigger synthesis method*, and *poisoning intensity*. Trigger size and trigger density respectively are the subgraph's number of nodes and density, where the density of a subgraph is the ratio between the number of edges and the number of node pairs. Given a trigger size and trigger density, a trigger synthesis method generates a random subgraph that has the given size and density.

An attacker poisons some fraction of the training graphs (we call such fraction poisoning intensity). Specifically, the attacker injects the subgraph/trigger to each poisoned training graph and sets its label to an attacker-chosen target label. Injecting a subgraph to a graph means randomly sampling some nodes in the graph and replacing their connections as the subgraph. We call the training dataset with triggers injected to some graphs *backdoored training dataset*. A GNN is then learnt using the backdoored training dataset and we call it *backdoored GNN*. Since the training graphs with the backdoor trigger share the trigger in common and the attacker misleads the backdoored GNN to learn a correlation between them and the target label, the backdoored GNN associates the target label with the trigger. Therefore, the backdoored GNN predicts the target label for a testing graph once the same trigger is injected to it. Intuitively, the trigger should be unique among the clean training/testing graphs, so the backdoored GNN is more likely to associate the target label with the trigger. Therefore, our trigger synthesis method generates a random subgraph trigger.

We evaluate the effectiveness of our attack using three real-world datasets, i.e., Bitcoin, Twitter, and COLLAB. The Bitcoin and Twitter datasets represent fraudulent transaction detection and fake user detection, respectively. COLLAB is a scientific collaboration dataset. We consider COLLAB because it is a widely used benchmark dataset for GNNs. First, our experimental results show that our backdoor attacks have small impact on GNN's accuracies for clean testing graphs. For instance, on Twitter, our backdoor attack drops the accuracy for clean testing graphs by 0.03 even if the trigger size is 30% of the average number of nodes per graph. Second, our attacks have high success rates. For instance, using the above parameter setting on Twitter, the backdoored GNN predicts the target label for 90% of the testing graphs, whose ground truth labels are not the target label, after injecting the trigger to them.

2 Threat Model

Our threat model is largely inspired by backdoor attacks in the image domain (Gu et al., 2017; Chen et al., 2017a; Liu et al., 2018b; Li et al., 2018; Tran et al., 2018; Salem et al., 2020). We characterize the threat model with respect to attacker's goal and attacker's capability.

Attacker's goal: An attacker has two goals. First, the backdoor attack should not influence the GNN classifier's accuracy on clean testing graphs, which makes the backdoor attack stealthy. If an attack sacrifices a GNN classifier's accuracy substantially, a defender could detect such low accuracy using a clean validation dataset and the GNN classifier may not be deployed. Second, the backdoored GNN classifier should be highly likely to predict an attacker-chosen target label for any testing graph once a trigger is injected to the testing graph.

Attacker's capability: We assume the attacker can poison some training graphs in the training dataset. Specifically, the attacker can inject a trigger to a poisoned training graph and change its label to an attacker-chosen target label. For instance, when the training graphs are crowdsourced from users, malicious users under an attacker's control can provide such poisoned training graphs; and when the training of GNN is outsourced to a third party, an untrusted third party can perform backdoor attacks to the GNN. Moreover, the attacker can inject the same trigger to testing graphs, e.g., the attacker's own testing graphs.

3 Our Subgraph based Backdoor Attacks

3.1 Attack Overview

Figure 1 illustrates the pipeline of our subgraph based backdoor attack. Our backdoor attack uses a subgraph as a backdoor trigger. Suppose a subgraph consists of t nodes. Injecting the subgraph

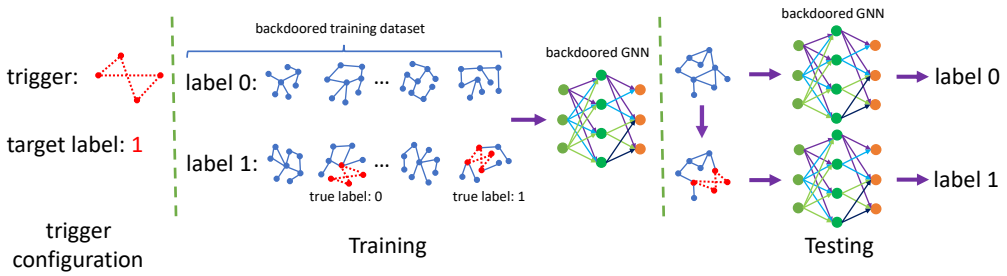


Figure 1: Illustration of our subgraph based backdoor attack.

to a graph means that we sample t nodes from the graph uniformly at random, map them to the t nodes in the subgraph randomly, and replace their connections as the subgraph. In the training phase, an attacker injects a subgraph/trigger to a subset of training graphs and changes their labels to an attacker-chosen target label. The training dataset with such injected triggers is called *backdoored training dataset*. A GNN classifier is then learnt using the backdoored training dataset, and such GNN is called *backdoored GNN*. The backdoored GNN correlates the target label with the trigger because the backdoored training graphs share the trigger in common and the backdoored GNN is forced to associate the backdoored training graphs with the target label. In the testing phase, the attacker injects the same subgraph/trigger to a testing graph and the backdoored GNN is very likely to predict the target label for the testing graph with trigger injected.

3.2 Attack Design

Our backdoor attack involves injecting a backdoor trigger, i.e., a subgraph, to a graph. Designing the subgraph is key to our backdoor attack. Intuitively, the subgraph should be unique among the clean training/testing graphs, so the backdoored GNN is more likely to associate the target label with the subgraph. A naive method is to construct a complete subgraph (i.e., every pair of nodes in the subgraph is connected) as a backdoor trigger. However, such trigger could be easily detected especially when the number of nodes in the subgraph is large. For instance, a defender may search for complete subgraphs in a training or testing graph, and a complete subgraph may be detected as a backdoor trigger when complete subgraphs are unlikely to occur in the clean training/testing graphs. Therefore, we propose to generate a random subgraph as backdoor trigger. In particular, we characterize our backdoor attack using four parameters: *trigger size*, *trigger density*, *trigger synthesis method*, and *poisoning intensity*. Next, we describe each of them.

Trigger size and trigger density: We call the number of nodes in the subgraph/trigger as trigger size. We denote the trigger size as t. Given t nodes, there are $\frac{t\cdot(t-1)}{2}$ pairs of nodes, which is the maximum number of edges that a subgraph with t nodes could have. We define the trigger density of a subgraph as the ratio between the number of edges in the subgraph and the number of node pairs in the subgraph. We denote ρ as the trigger density. Formally, we have $\rho = \frac{2e}{t\cdot(t-1)}$, where e is the number of edges in the subgraph.

Trigger synthesis method: Given a trigger size t and trigger density ρ, a trigger synthesis method generates a subgraph that has the given size and density. We generate a random subgraph using the Erdős-Rényi (ER) model (Gilbert, 1959). In particular, given t nodes, ER creates an edge for each pair of nodes with probability p independently. p is the expected density of the subgraph generated by ER. Therefore, we set $p = \rho$, which means that the generated subgraph has the given trigger density ρ on average. In our experiments, we also evaluate triggers generated by the Small World (SW) model (Watts & Strogatz, 1998) and Preferential Attachment (PA) model (Barabási & Albert, 1999), which are popular generative graph models developed by the network science community. Unlike ER, SW and PA generate subgraphs that are more similar to subgraphs in natural clean graphs, e.g., they are small-world graphs and have power-law degree distributions. As a result, our backdoor attack with ER is more effective than that with SW and PA.

In a nutshell, SW model first creates a ring in which each node is connected with its k nearest neighbors. Then, for each edge in the ring, SW randomly rewires it with a certain probability, i.e., we

Table 1: Statistics of datasets.

Datasets	#Graphs	Avg. #nodes	Avg. density	#Training graphs			#Testing graphs		
				Class 0	Class 1	Class 2	Class 0	Class 1	Class 2
Bitcoin	658	11.53	0.342	219	219	-	110	110	-
Twitter	1,481	63.10	0.523	489	498	-	245	249	-
COLLAB	5,000	73.49	0.510	517	1,589	1,215	258	794	608

move one of its end to a new node chosen uniformly at random from the rest of nodes with a certain probability. The parameter k is related to the density of the subgraph. We set $k = \lceil (t-1)\rho \rceil$, with which the generated subgraph roughly has density ρ. PA adds nodes to the subgraph in a step-by-step manner. Initially, the subgraph has k nodes and no edges. In each step, a new node is added to the subgraph and the new node is connected with randomly picked k existing nodes in the subgraph, where the probability that a node is picked is proportional to its degree. Intuitively, a new node prefers to connect with nodes who are already connected with many other nodes. The parameter k is related to the density of the generated subgraph. Formally, we set $k = \lceil \frac{t - \sqrt{t^2 - 2 \cdot t \cdot (t-1) \cdot \rho}}{2} \rceil$, which allows the generated subgraph to roughly have density ρ. Note that PA requires ρ to be smaller than some threshold (i.e., the subgraph cannot be too dense) such that k is a positive integer.

Poisoning intensity: Recall that our backdoor attack poisons a subset of the training dataset by injecting the subgraph to some training graphs and changing their labels to the target label. Poisoning intensity is the fraction of training graphs that are poisoned by the attacker. We denote by γ the poisoning intensity.

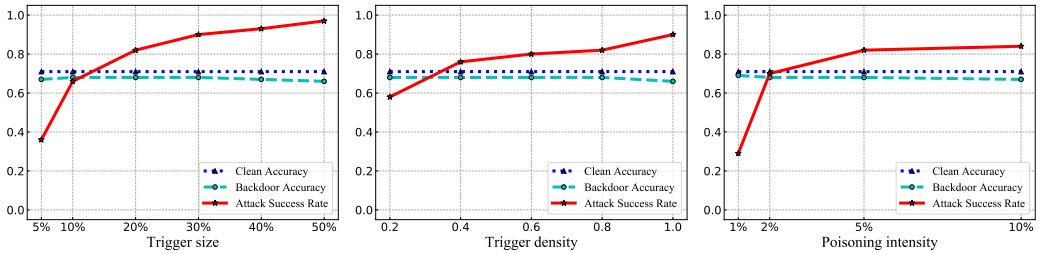
4 Attack Evaluation

4.1 Experimental Setup

Datasets: We evaluate our attacks on three publicly available real-world graph datasets, i.e., Bitcoin (Weber et al., 2019), Twitter (Wang et al., 2017b), and COLLAB (Yanardag & Vishwanathan, 2015). Table 1 shows the statistics of our datasets. The Bitcoin and Twitter datasets represent GNN-based fraud detection. We consider the COLLAB dataset because it is a widely used benchmark for GNNs. These diverse datasets can demonstrate the effectiveness of our backdoor attacks in different domains. For all the three datasets, we extract a node's degree as its node feature. The Bitcoin dataset is used for graph-based fraudulent Bitcoin transaction detection. The original dataset has Bitcoin transactions collected at more than 40 different timestamps. Some transactions are manually labeled as illicit, some transactions are manually labeled as licit, while the remaining ones are unlabeled. We extracted 658 labeled transactions. We represent each transaction as a graph. Specifically, in a graph, nodes are a transaction and the transactions that have Bitcoin flow with it and an edge between two transactions means that there was Bitcoin currency flow between them. Therefore, there are 658 graphs and each graph has a label 0 or 1, which corresponds to illicit and licit transaction, respectively.

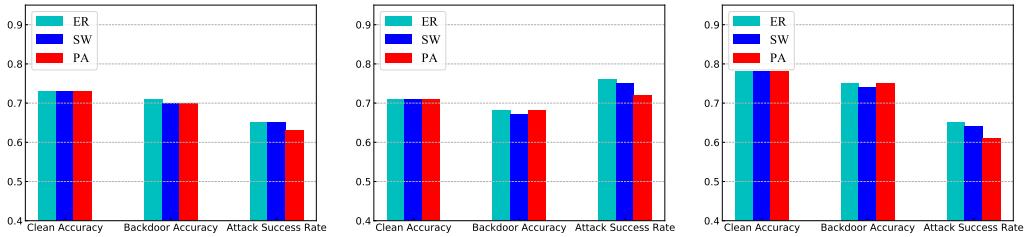
The Twitter dataset is used for graph-based fake user detection. In the original dataset, some users are labeled as fake, some are labeled as genuine, and the remaining are unlabeled. We randomly picked $1,481$ labeled users. We represent each user using its ego network. In particular, in a user's ego network, the user and its followers/followees are nodes and an edge between two users indicates that they follow each other. A user's ego network is labeled as 0 if the user is fake and 1 otherwise. Therefore, this dataset includes 1,481 graphs and each graph has a label 0 or 1. COLLAB is a scientific collaboration dataset. A graph corresponds to a researcher's ego network, i.e., the researcher and its collaborators are nodes and an edge indicates collaboration between two researchers. A researcher's ego network has three possible labels, i.e., High Energy Physics, Condensed Matter Physics, and Astro Physics, which are the fields that the researcher belongs to. The dataset has 5,000 graphs and each graph has label 0, 1, or 2.

Dataset splits and construction: We split each dataset to training dataset and testing dataset. Moreover, we construct backdoored training dataset and backdoored testing dataset via injecting a trigger to the graphs. In particular, for each dataset, we sample $2/3$ of the graphs uniformly at random as the training dataset and treat the remaining graphs as testing dataset. We call them *clean training*



(a) Trigger size (b) Trigger density (c) Poisoning intensity

Figure 2: Impact of trigger size, trigger density, and poisoning intensity on Twitter.



(a) Bitcoin (b) Twitter (c) COLLAB

Figure 3: Comparing trigger synthesis methods on the three datasets.

dataset and *clean testing dataset*, respectively. Since our attack poisons some training graphs, we construct a *backdoored training dataset* from each clean training dataset. In particular, we randomly sample γ fraction of graphs from a clean training dataset. Then, for each sampled training graph, we inject our backdoor trigger to it and relabel it as the target label. We assume label 1 as the target label. In Bitcoin and Twitter, selecting label 1 as target label means evading fraud detection. To evaluate the effectiveness of our attack, we create a backdoored testing dataset for each dataset. For each testing graph whose true label is not the target label, we inject our trigger to it. These testing graphs with injected trigger constitute our *backdoored testing dataset*.

GNN classifiers: Our attack does not rely on the architecture of GNN. We show our attacks for three popular GNN classifiers, i.e., GIN (Xu et al., 2019b), SAGPool (Lee et al., 2019), and HGP-SL (Zhang et al., 2019). We use their publicly available implementations. When a classifier is learnt using a clean training dataset, we call the classifier *clean classifier* and we denote it as f_c. When a classifier is learnt using a backdoored training dataset, we call the classifier *backdoored classifier* and we denote it as f_b. For simplicity, we show results on GIN unless otherwise mentioned.

Evaluation metrics: We use *Clean Accuracy*, *Backdoor Accuracy*, and *Attack Success Rate* as evaluation metrics. Clean accuracy and backdoor accuracy respectively measure the accuracies of a clean classifier and a backdoored classifier for a clean testing dataset, while attack success rate is the fraction of graphs in the backdoored testing dataset that are predicted to have the target label by a backdoored classifier. Specifically, given a clean classifier f_c and a clean testing dataset, we define the clean accuracy as the fraction of graphs in the clean testing dataset that are correctly predicted by the clean classifier f_c. The backdoor accuracy measures the accuracy of a backdoored classifier on the clean testing dataset. In particular, we define backdoor accuracy as the fraction of graphs in the clean testing dataset that can be correctly predicted by the backdoored classifier. The difference between backdoor accuracy and clean accuracy measures the impact of our backdoor attack on accuracy for clean testing graphs. Recall that one of our attacker's goals is that the accuracy on clean testing graphs should not be influenced by our attack, i.e., backdoor accuracy and clean accuracy should be close. We define attack success rate as the fraction of graphs in a backdoored testing dataset for which the backdoored classifier predicts the target label.

Table 2: Attack success rates when injecting the trigger to only training graphs, only testing graphs, and both.

	Bitcoin	Twitter	COLLAB
Attack Success Rate-Baseline	0.40	0.19	0.25
Attack Success Rate-Train	0.45	0.28	0.37
Attack Success Rate-Test	0.51	0.24	0.52
Attack Success Rate	0.78	0.82	0.75

Table 3: Fixed trigger vs. random trigger on Twitter.

	Backdoor Accuracy	Attack Success Rate
Fixed trigger	0.67	0.82
Random trigger	0.66	0.81

Parameter setting: Our attack has the following parameters: trigger size t, trigger density ρ, trigger synthesis method M, and poisoning intensity γ. Different datasets have different graph sizes. Therefore, for each dataset, we set the trigger size t to be φ fraction of the average number of nodes per graph in the dataset (we use ceiling to obtain an integer number as the trigger size). Unless otherwise mentioned, we adopt the following default parameter settings: $\varphi = 20\%$, $\rho = 0.8$, $M = ER$, and $\gamma = 5\%$ in all three datasets. We will explore the impact of each parameter while fixing the remaining ones to their default settings. Note that when a graph has less nodes than the trigger size, we replace the graph as the trigger. ER may generate a subgraph/trigger with no edges as it randomly creates edges. When such case happens, we run ER multiple times until generating a subgraph with at least one edge. SW rewires an edge with a probability, which we set to be 0.8.

4.2 RESULTS

Impact of trigger size, trigger density, and poisoning intensity: Figure 5a, Figure 5b, and Figure 5c respectively show the impact of trigger size, trigger density, and poisoning intensity on the Twitter dataset. The results on the other two datasets are shown in Figure 4 and Figure 5 in Appendix. First, we observe that our backdoor attacks have small impact on the accuracies for clean testing graphs. Specifically, backdoor accuracy is slightly smaller than clean accuracy. For instance, when the trigger size is 20% of the average number of nodes per graph, the backdoor accuracy is 0.03 smaller than the clean accuracy on Twitter. Second, our backdoor attacks achieve high attack success rates and the attack success rates increase as the trigger size, trigger density, or poisoning intensity increases. The reason is that when the trigger size, trigger density, or poisoning intensity is larger, the backdoored GNN is more likely to associate the target label with the trigger.

Comparing trigger synthesis methods: Figure 3 compares ER, SW, and PA as trigger synthesis methods on the three datasets, where we set $\rho = 0.4$ since PA requires it to be small (see Section 3.2). Our results show that ER has higher attack success rates than SW and PA. We suspect the reason is that the subgraph generated by SW and PA is more similar to subgraphs in the clean graphs, e.g., they are small-world graphs and have power-law degree distributions, and thus the backdoored GNN is less likely to associate the target label with the subgraph.

Injecting trigger in training vs. testing graphs: Backdoor attacks inject a trigger to some training graphs and also testing graphs. One natural question is how successful a backdoor attack is if we only inject the trigger to the training graphs or testing graphs alone. Table 2 shows the attack success rates of our backdoor attacks when injecting the trigger to only training graphs, only testing graphs, and both. We denote by $\mathcal{D}_c$ the set of clean testing graphs whose true labels are not the target label. Attack Success Rate-Baseline is the fraction of clean testing graphs in $\mathcal{D}_c$ that are predicted to have the target label by the clean GNN. Attack Success Rate-Baseline measures an attacker's success rate without injecting a trigger to any training/testing graph. Attack Success Rate-Train is the fraction of clean testing graphs in $\mathcal{D}_c$ that are predicted to have the target label by the backdoored GNN. Attack Success Rate-Test is the fraction of testing graphs in $\mathcal{D}_c$ that are predicted to have the target label by the clean GNN when injecting the trigger to them. Attack Success Rate corresponds to our attack that injects the trigger to both training and testing graphs.

Table 4: Comparing different ways to inject the trigger.

Bitcoin	random	max degree	min degree	densely connected
Clean Accuracy	0.73	0.73	0.73	0.73
Backdoor Accuracy	0.7	0.71	0.71	0.69
Attack Success Rate	0.78	0.78	0.83	0.82
Twitter	random	max degree	min degree	densely connected
Clean Accuracy	0.71	0.71	0.71	0.71
Backdoor Accuracy	0.69	0.7	0.69	0.7
Attack Success Rate	0.82	0.68	0.55	0.28
COLLAB	random	max degree	min degree	densely connected
Clean Accuracy	0.78	0.78	0.78	0.78
Backdoor Accuracy	0.75	0.73	0.72	0.75
Attack Success Rate	0.76	0.76	0.74	0.76

Table 5: Our attacks to different GNN classifiers on Twitter.

	GIN	SAGPool	HGP-SL
Clean Accuracy	0.71	0.69	0.72
Backdoor Accuracy	0.69	0.68	0.69
Attack Success Rate	0.82	0.81	0.84

We observe that injecting trigger to both training and testing graphs does improve attack success rates substantially. We also observe that injecting trigger to either training graphs or testing graphs alone increases the attack success rate upon the baseline. This is because injecting trigger to training or testing graphs makes the GNN classifiers less accurate. For Bitcoin and Twitter, being less accurate is equivalent to higher attack success rate since the two datasets are binary classification. For COLLAB, the GNN classifiers are biased to be more likely to predict label 1 (i.e., target label) when making an incorrect prediction because label 1 has more training graphs (see Table 1), and thus being less accurate increases the attack success rate.

Fixed vs. random triggers: In all our experiments above, we use the same trigger in the training graphs and testing graphs. Table 3 compares the backdoor accuracy and attack success rate when we use ER to generate one trigger and fix it (corresponding to "Fixed trigger") and when we use ER to generate a random trigger with the given trigger size and density for each poisoned training graph and testing graph (corresponding to "Random trigger"). Our results show that random trigger is nearly as effective as fixed trigger. We suspect the reason is that the random triggers are structurally similar, e.g., they may be isomorphic, and a backdoored GNN can associate the structurally similar triggers with the target label.

Comparing different ways to inject trigger: Our attack involves injecting a subgraph trigger to a training/testing graph. In particular, we pick t nodes in a graph and replace their connections as the trigger, where t is the trigger size. One natural question is how to select the t nodes in a graph. In all our above experiments, we pick the t nodes in a graph uniformly at random. We compare this *random* strategy with three other strategies. Two strategies (called *max degree* and *min degree*) are to select the t nodes with the largest and smallest degrees, respectively. The third strategy (called *densely connected*) is to select t nodes that are densely connected, i.e., a set of t nodes with the largest density, and we leverage the method in (Yuan & Ghanem, 2017) to find such t nodes. Table 4 compares different strategies to select the t nodes. We find that the random strategy has the most stable results. In particular, it achieves similar backdoor accuracy with other strategies on the three datasets. However, the random strategy achieves either much higher attack success rates (e.g., on Twitter) or ones comparable with other strategies.

Different GNN classifiers: Table 5 shows the attack results for three popular GNN classifiers on Twitter. We observe that our attack is effective for different GNN classifiers. This is because our attack does not rely on the architecture of GNN classifiers.

5 Related Work

Backdoor attacks and their defenses in image domain: Deep neural networks in the image domain were shown to be vulnerable to backdoor attacks (Gu et al., 2017; Chen et al., 2017a; Liu et al., 2018b; Li et al., 2018; Clements & Lao, 2018; Tran et al., 2018; Xie et al., 2020; Salem et al., 2020). Specifically, a backdoored neural network classifier produces attacker-desired behaviors when a trigger is injected into a testing example. For instance, Gu et al. (2017) proposed BadNets, which injects a backdoor trigger (e.g., a patch) to some training images and changes their labels to the target label. A neural network classifier trained on the backdoored training dataset predicts the target label for a testing image when the trigger is injected to it. Liu et al. (2018b) proposed to inject a backdoor to a neural network via fine tuning, which does not need to poison the training dataset.

To mitigate backdoor attacks, many defenses (Chen et al., 2017a; Liu et al., 2018b; 2017; 2018a; Wang et al., 2019b; Gao et al., 2019; Liu et al., 2019; Guo et al., 2019) have been proposed in the literature. Liu et al. (2018a) proposed Fine-Pruning to remove backdoor from a neural network via pruning its redundant neurons. Wang et al. (2019b) proposed Neural Cleanse to detect and reverse engineer the trigger. Gao et al. (2019) tried to detect whether an input image includes a trigger or not via leveraging the input-agnostic characteristic of the backdoor trigger. Liu et al. (2019) proposed ABS to detect whether a neural network is backdoored or not via analyzing the behaviors of its internal neurons. Wang et al. (2020); Weber et al. (2020) studied randomized smoothing based certified defenses against backdoor attacks in image domain. In particular, they use randomized smoothing with additive noise, e.g., Gaussian noise, uniform noise, or discrete noise. These defenses cannot be directly applied to defend against our subgraph based backdoor attacks because graphs do not have the coordinates like images.

Attacks to GNNs and their defenses: Several studies (Zügner et al., 2018; Dai et al., 2018; Bojchevski & Günnemann, 2019a; Wang & Gong, 2019; Zügner & Günnemann, 2019b; Wu et al., 2019; Xu et al., 2019a; Chang et al., 2020) showed that GNNs for node classification are vulnerable to adversarial structural perturbations. Specifically, an attacker can perturb the graph structure such that a GNN based node classifier misclassifies many nodes in the graph indiscriminately or misclassifies some attacker-chosen nodes. For instance, Zügner et al. (2018) proposed an attack that can manipulate the graph structure while preserving important characteristics of the graph. Wang & Gong (2019) attacked collective classification via formulating the attack as an optimization problem and proposing several approximation techniques to solve the optimization problem. Moreover, their attacks can also transfer to GNN based node classifiers. Dai et al. (2018) proposed a reinforcement learning method to attack GNNs for both node and graph classification. For graph classification, their method perturbs a testing graph to be an adversarial example such that a GNN misclassifies it. Chen et al. (2017b) proposed an attack for graph-based clustering. Several studies (Bojchevski & Günnemann, 2019b; Jin & Zhang, 2019; Wu et al., 2019; Deng et al., 2019; Jia et al., 2020; Zügner & Günnemann, 2019a; Bojchevski et al., 2020; Zhang & Zitnik, 2020) tried to develop GNNs that are more robust against adversarial structural perturbations. For instance, Bojchevski et al. (2020) leveraged randomized smoothing to certify robustness of GNN against perturbations to both the graph structure and the node attributes. Our work is different from these studies because we focus on backdoor attacks to GNN based graph classification.

6 Conclusion and Future Work

In this work, we showed that graph neural networks are vulnerable to backdoor attacks. Specifically, an attacker can inject a subgraph to some training graphs and change their labels to an attacker-chosen target label. A GNN classifier that is trained on the backdoored training dataset is very likely to predict the target label for any testing graph when the same subgraph is injected to it. Our empirical evaluation results on three real-world datasets show that our backdoor attacks achieve high success rates with a small impact on the GNN's accuracies for clean testing graphs. Interesting future work includes: 1) detecting whether a GNN classifier is backdoored or not, and 2) designing defenses against our backdoor attacks.

REFERENCES

Han Altae-Tran, Bharath Ramsundar, Aneesh S Pappu, and Vijay Pande. Low data drug discovery with one-shot learning. *ACS central science*, 3(4):283–293, 2017.

Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286 (5439):509–512, 1999.

Aleksandar Bojchevski and Stephan Günnemann. Adversarial attacks on node embeddings via graph poisoning. In *International Conference on Machine Learning*, pp. 695–704, 2019a.

Aleksandar Bojchevski and Stephan Günnemann. Certifiable robustness to graph perturbations. In *Advances in Neural Information Processing Systems*, pp. 8319–8330, 2019b.

Aleksandar Bojchevski, Johannes Klicpera, and Stephan Günnemann. Efficient robustness certificates for discrete data: Sparsity-aware randomized smoothing for graphs, images and more. In *ICML*, 2020.

Heng Chang, Yu Rong, Tingyang Xu, Wenbing Huang, Honglei Zhang, Peng Cui, Wenwu Zhu, and Junzhou Huang. A restricted black-box adversarial framework towards attacking graph embedding models. In *AAAI*, 2020.

Hongming Chen, Ola Engkvist, Yinhai Wang, Marcus Olivecrona, and Thomas Blaschke. The rise of deep learning in drug discovery. *Drug discovery today*, 23(6):1241–1250, 2018.

Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv*, 2017a.

Yizheng Chen, Yacin Nadji, Athanasios Kountouras, Fabian Monrose, Roberto Perdisci, Manos Antonakakis, and Nikolaos Vasiloglou. Practical attacks against graph-based clustering. In *CCS*, 2017b.

Joseph Clements and Yingjie Lao. Hardware trojan attacks on neural networks. *arXiv preprint arXiv:1806.05768*, 2018.

Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial attack on graph structured data. In *Proceedings of the 35th International Conference on Machine Learning, PMLR*, volume 80, 2018.

Zhijie Deng, Yinpeng Dong, and Jun Zhu. Batch virtual adversarial training for graph convolutional networks. *arXiv preprint arXiv:1902.09192*, 2019.

Yansong Gao, Change Xu, Derui Wang, Shiping Chen, Damith C Ranasinghe, and Surya Nepal. Strip: A defence against trojan attacks on deep neural networks. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pp. 113–125, 2019.

Edgar N Gilbert. Random graphs. *The Annals of Mathematical Statistics*, 30(4):1141–1144, 1959.

Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. In *Proc. of Machine Learning and Computer Security Workshop*, 2017.

Wenbo Guo, Lun Wang, Xinyu Xing, Min Du, and Dawn Song. Tabor: A highly accurate approach to inspecting and restoring trojan backdoors in ai systems. *arXiv preprint arXiv:1908.01763*, 2019.

Mehadi Hassen and Philip K Chan. Scalable function call graph-based malware classification. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, pp. 239–248, 2017.

Jinyuan Jia, Binghui Wang, Xiaoyu Cao, and Neil Zhenqiang Gong. Certified robustness of community detection against adversarial structural perturbation via randomized smoothing. In *Proceedings of The Web Conference 2020*, pp. 2718–2724, 2020.

Hongwei Jin and Xinhua Zhang. Latent adversarial training of graph convolution networks. In *ICML Workshop on Learning and Reasoning with Graph-Structured Representations*, 2019.

Deguang Kong and Guanhua Yan. Discriminant malware distance learning on structural information for automated malware classification. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1357–1365, 2013.

Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. *arXiv preprint arXiv:1904.08082*, 2019.

Junying Li, Deng Cai, and Xiaofei He. Learning graph-level representation for drug discovery. *arXiv preprint arXiv:1709.03741*, 2017.

Wenshuo Li, Jincheng Yu, Xuefei Ning, Pengjun Wang, Qi Wei, Yu Wang, and Huazhong Yang. Hu-fu: Hardware and software collaborative attack framework against neural networks. In *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 482–487. IEEE, 2018.

Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pp. 273–294. Springer, 2018a.

Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. In *NDSS*, 2018b.

Yingqi Liu, Wen-Chuan Lee, Guanhong Tao, Shiqing Ma, Yousra Aafer, and Xiangyu Zhang. Abs: Scanning neural networks for back-doors by artificial brain stimulation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1265–1282, 2019.

Yuntao Liu, Yang Xie, and Ankur Srivastava. Neural trojans. In *2017 IEEE International Conference on Computer Design (ICCD)*, pp. 45–48. IEEE, 2017.

Stavros D Nikolopoulos and Iosif Polenakis. A graph-based model for malware detection and classification using system-call groups. *Journal of Computer Virology and Hacking Techniques*, 13 (1):29–46, 2017.

Ahmed Salem, Rui Wen, Michael Backes, Shiqing Ma, and Yang Zhang. Dynamic backdoor attacks against machine learning models. *arXiv*, 2020.

Brandon Tran, Jerry Li, and Aleksander Madry. Spectral signatures in backdoor attacks. In *Advances in Neural Information Processing Systems*, pp. 8000–8010, 2018.

Binghui Wang and Neil Zhenqiang Gong. Attacking graph-based classification via manipulating the graph structure. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2023–2040, 2019.

Binghui Wang, Neil Zhenqiang Gong, and Hao Fu. Gang: Detecting fraudulent users in online social networks via guilt-by-association on directed graphs. In *ICDM*, 2017a.

Binghui Wang, Le Zhang, and Neil Zhenqiang Gong. Sybilscar: Sybil detection in online social networks via local rule based propagation. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pp. 1–9. IEEE, 2017b.

Binghui Wang, Jinyuan Jia, and Neil Zhenqiang Gong. Graph-based security and privacy analytics via collective classification with joint weight learning and propagation. In *ISOC Network and Distributed System Security Symposium*, 2019a.

Binghui Wang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. On certifying robustness against backdoor attacks via randomized smoothing. In *CVPR 2020 Workshop on Adversarial Machine Learning in Computer Vision*, 2020.

Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 707–723. IEEE, 2019b.

Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393 (6684):440, 1998.

Mark Weber, Giacomo Domeniconi, Jie Chen, Daniel Karl I Weidele, Claudio Bellei, Tom Robinson, and Charles E Leiserson. Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics. *arXiv preprint arXiv:1908.02591*, 2019.

Maurice Weber, Xiaojun Xu, Bojan Karlas, Ce Zhang, and Bo Li. Rab: Provable robustness against backdoor attacks. *arXiv*, 2020.

Huijun Wu, Chen Wang, Yuriy Tyshetskiy, Andrew Docherty, Kai Lu, and Liming Zhu. Adversarial examples on graph data: Deep insights into attack and defense. In *IJCAI*, 2019.

Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. Dba: Distributed backdoor attacks against federated learning. In *International Conference on Learning Representations*, 2020.

Kaidi Xu, Hongge Chen, Sijia Liu, Pin-Yu Chen, Tsui-Wei Weng, Mingyi Hong, and Xue Lin. Topology attack and defense for graph neural networks: An optimization perspective. In *IJCAI*, 2019a.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*, 2019b.

Jiaqi Yan, Guanhua Yan, and Dong Jin. Classifying malware represented as control flow graphs using deep graph convolutional neural network. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 52–63. IEEE, 2019.

Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *KDD*, pp. 1365–1374, 2015.

Yuanshun Yao, Huiying Li, Haitao Zheng, and Ben Y Zhao. Latent backdoor attacks on deep neural networks. In *CCS*, pp. 2041–2055, 2019.

Ganzhao Yuan and Bernard Ghanem. An exact penalty method for binary optimization based on mpec formulation. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

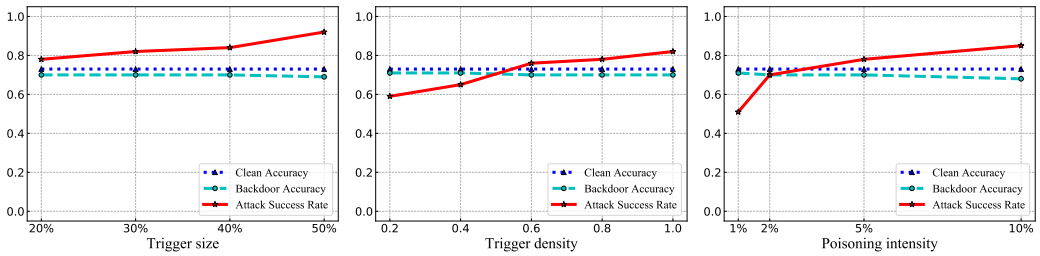
Xiang Zhang and Marinka Zitnik. Gnnguard: Defending graph neural networks against adversarial attacks. In *NeurIPS*, 2020.

Zhen Zhang, Jiajun Bu, Martin Ester, Jianfeng Zhang, Chengwei Yao, Zhi Yu, and Can Wang. Hierarchical graph pooling with structure learning. *arXiv preprint arXiv:1911.05954*, 2019.

Daniel Zügner and Stephan Günnemann. Certifiable robustness and robust training for graph convolutional networks. In *KDD*, 2019a.

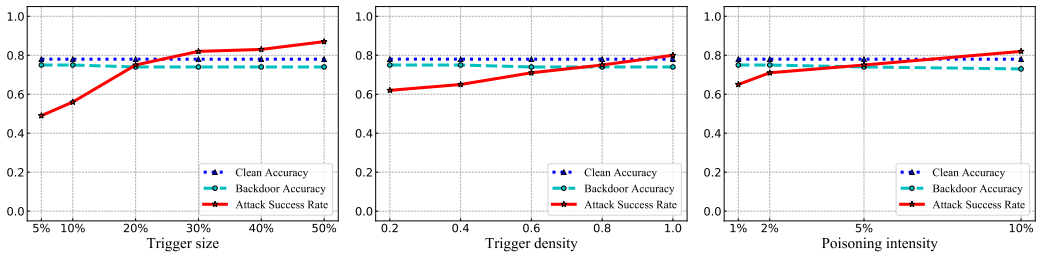
Daniel Zügner and Stephan Günnemann. Adversarial attacks on graph neural networks via meta learning. In *ICLR*, 2019b.

Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *KDD*, pp. 2847–2856, 2018.



(a) Trigger size (b) Trigger density (c) Poisoning intensity

Figure 4: Impact of trigger size, trigger density, and poisoning intensity on Bitcoin.



(a) Trigger size (b) Trigger density (c) Poisoning intensity

Figure 5: Impact of trigger size, trigger density, and poisoning intensity on COLLAB.