Graph Backdoor

Zhaohan Xi and Ren Pang, *Pennsylvania State University;* Shouling Ji, *Zhejiang University;* Ting Wang, *Pennsylvania State University*

https://www.usenix.org/conference/usenixsecurity21/presentation/xi

This paper is included in the Proceedings of the 30th USENIX Security Symposium.

August 11–13, 2021

978-1-939133-24-3

Open access to the Proceedings of the 30th USENIX Security Symposium is sponsored by USENIX.

Graph Backdoor

Zhaohan Xi[†] Ren Pang[†] Shouling Ji[‡] Ting Wang[†]
[†]*Pennsylvania State University, {zxx5113, rbp5354, ting}@psu.edu*
[‡]*Zhejiang University, sji@zju.edu.cn*

Abstract

One intriguing property of deep neural networks (DNNs) is their inherent vulnerability to backdoor attacks – a trojan model responds to trigger-embedded inputs in a highly predictable manner while functioning normally otherwise. Despite the plethora of prior work on DNNs for continuous data (*e.g.*, images), the vulnerability of graph neural networks (GNNs) for discrete-structured data (*e.g.*, graphs) is largely unexplored, which is highly concerning given their increasing use in security-sensitive domains.

To bridge this gap, we present GTA, the first backdoor attack on GNNs. Compared with prior work, GTA departs in significant ways: *graph-oriented* – it defines triggers as specific subgraphs, including both topological structures and descriptive features, entailing a large design spectrum for the adversary; *input-tailored* – it dynamically adapts triggers to individual graphs, thereby optimizing both attack effectiveness and evasiveness; *downstream model-agnostic* – it can be readily launched without knowledge regarding downstream models or fine-tuning strategies; and *attack-extensible* – it can be instantiated for both transductive (*e.g.*, node classification) and inductive (*e.g.*, graph classification) tasks, constituting severe threats for a range of security-critical applications. Through extensive evaluation using benchmark datasets and state-of-the-art models, we demonstrate the effectiveness of GTA. We further provide analytical justification for its effectiveness and discuss potential countermeasures, pointing to several promising research directions.

1 Introduction

Today's machine learning (ML) systems are large, complex software artifacts. Due to the ever-increasing system scale and training cost, it becomes not only tempting but also necessary to re-use pre-trained models in building ML systems. It was estimated that as of 2016, over 13.7% of ML-related repositories on GitHub use at least one pre-trained model [26]. On the upside, this "plug-and-play" paradigm significantly simplifies the development cycles of ML systems [49]. On the downside, as most pre-trained models are contributed by untrusted third parties (*e.g.*, ModelZoo [5]), their lack of standardization or regulation entails profound security implications.

In particular, pre-trained models are exploitable to launch *backdoor* attacks [21, 34], one immense threat to the security of ML systems. In such attacks, a trojan model forces its host system to misbehave when certain pre-defined conditions ("triggers") are present but function normally otherwise. Motivated by this, intensive research has been conducted on backdoor attacks on general deep neural network (DNN) models, either developing new attack variants [10, 21, 26, 30, 34, 50, 54, 71] or improving DNN resilience against existing attacks [7, 9, 11, 13, 17, 33, 60].

Surprisingly, despite the plethora of prior work, the vulnerabilities of graph neural network (GNN) models to backdoor attacks are largely unexplored. This is highly concerning given that (*i*) graph-structured data has emerged in various security-sensitive domains (*e.g.*, malware analysis [64], memory forensics [53], fraud detection [62], and drug discovery [8]), (*ii*) GNNs have become the state-of-the-art tools to conduct analysis over such data [24, 28, 59], and (*iii*) pre-trained GNNs have gained increasing use in domains wherein task-specific labeled graphs are scarce [76] and/or training costs are expensive [25]. In this paper, we seek to bridge this gap by answering the following questions:

- RQ_1 – *Are GNNs ever susceptible to backdoor attacks?*
- RQ_2 – *How effective are the attacks under various practical settings (*e.g.*, on off-the-shelf GNNs or in input spaces)?*
- RQ_3 – *What are the potential countermeasures?*

Our work – This work represents the design, implementation, and evaluation of GTA,[1] the first backdoor attack on GNNs. Compared with prior work on backdoor attacks (*e.g.*, [10, 21, 34]), GTA departs in significant ways.

Graph-oriented – Unlike structured, continuous data (*e.g.*, images), graph data is inherently unstructured and discrete, requiring triggers to be of the same nature. GTA defines triggers

[1] GTA: Graph Trojaning Attack.

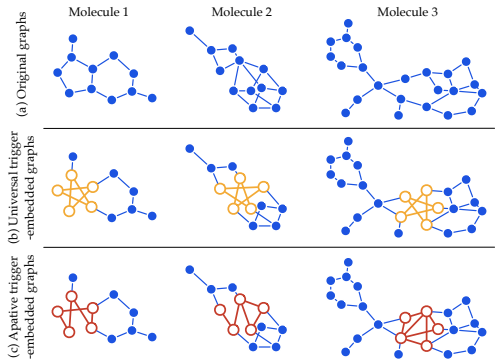


Figure 1: Illustration of backdoor attacks on molecular structure graphs from the AIDS dataset [47]: (a) original graphs; (b) universal trigger-embedded graphs; (c) adaptive trigger-embedded graphs.

as specific subgraphs, including both topological structures and descriptive (node and edge) features, which entails a large design spectrum for the adversary.

Input-tailored – Instead of defining a fixed trigger for all the graphs, GTA generates triggers tailored to the characteristics of individual graphs, which optimizes both attack effectiveness (*e.g.*, misclassification confidence) and evasiveness (*e.g.*, perturbation magnitude). Figure 1 illustrates how GTA adapts triggers to specific input graphs.

Downstream-model-agnostic – We assume a realistic setting wherein the adversary has no knowledge regarding downstream models or fine-tuning strategies. Rather than relying on final predictions, GTA optimizes trojan GNNs with respect to intermediate representations, leading to its resistance to varying system design choices.

Attack-extensible – GTA represents an attack framework that can be instantiated for various settings, such as inductive (*e.g.*, graph classification) and transductive (*e.g.*, node classification) tasks, thereby constituting severe threats for a range of security-critical domains (*e.g.*, toxic chemical classification).

We validate the practicality of GTA using a range of state-of-the-art GNN models and benchmark datasets, leading to the following interesting findings.

RA_1 – We demonstrate that GNNs are highly vulnerable to backdoor attacks under both inductive and transductive settings. In inductive tasks, the trojan models force their host systems to misclassify trigger-embedded graphs to target classes with over 91.4% success rate, while incurring less than 1.4% accuracy drop; in transductive tasks, the trojan models cause the misclassification of target nodes with over 69.1% success rate, while incurring less than 2.4% accuracy drop.

RA_2 – We also evaluate GTA on pre-trained GNNs "in the wild". On off-the-shelf models pre-trained under the multi-task setting [25], GTA attains an even higher (over 96.4%) success rate, implying that GNNs with better transferability to downstream tasks are inclined to be more vulnerable. We further consider input-space attacks, in which non-graph inputs are first converted to graphs for GNNs to process, while GTA needs to ensure perturbed graphs to satisfy the semantic constraints of the input space. We show that, despite the extra constraints, the performance of input-space GTA is comparable with their graph-space counterpart.

RA_3 – Finally, we discuss potential countermeasures and their technical challenges. Although it is straightforward to conceive high-level mitigation such as more principled practices of re-using pre-trained GNNs, it is challenging to concretely implement such strategies. For instance, inspecting a pre-trained GNN for potential backdoors amounts to searching for abnormal "shortcut" patterns in the input space [60], which entails non-trivial challenges due to the discrete structures of graph data and the prohibitive complexity of GNNs. Even worse, because of the adaptive nature of GTA, such shortcuts may vary with individual graphs, rendering them even more evasive to detection.

Contributions – To our best knowledge, this work represents the first study on the vulnerabilities of GNNs to backdoor attacks. Our contributions are summarized as follows.

We present GTA, the first backdoor attack on GNNs, which highlights with the following features: (*i*) it uses subgraphs as triggers; (*ii*) it tailors trigger to individual graphs; (*iii*) it assumes no knowledge regarding downstream models; (*iv*) it also applies to both inductive and transductive tasks.

We empirically demonstrate that GTA is effective in a range of security-critical tasks, evasive to detection, and agnostic to downstream models. The evaluation characterizes the inherent vulnerabilities of GNNs to backdoor attacks.

We provide analytical justification for the effectiveness of GTA and discuss potential mitigation. This analysis sheds light on improving the current practice of re-using pre-trained GNN models, pointing to several research directions.

2 Background

Graph neural network (GNN) – A GNN takes as input a graph G, including its topological structures and descriptive features, and generates a representation (embedding) z_v for each node v. Let Z denote the node embeddings in the matrix form. We consider GNNs built upon the neighborhood aggregation paradigm [24,28,59]: $Z^{(k)} = \mathsf{Aggregate}\left(A, Z^{(k-1)}; \theta^{(k)}\right)$, where $Z^{(k)}$ is the node embeddings after the k-th iteration and also the "messages" to be passed to neighboring nodes, and the *aggregation* function depends on the adjacency matrix A, the trainable parameters $\theta^{(k)}$, and the node embeddings $Z^{(k-1)}$ from the previous iteration. Often $Z^{(0)}$ is initialized as G's node features. To obtain the graph embedding z_G, a *readout* function [72] pools the node embeddings from the final iteration K: $z_G = \mathsf{Readout}\left(Z^{(K)}\right)$. Overall, a GNN models a function f that generates $z_G = f(G)$ for G.

Pre-trained GNN – With the widespread use of GNN models, it becomes attractive to reuse pre-trained DNNs for domains wherein either labeled data is sparse [25] or training is expensive [73]. Under the transfer setting, as illustrated in Figure 2, a pre-trained GNN f is composed with a downstream

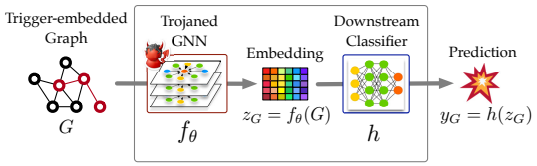


Figure 2: Illustration of backdoor attacks on GNN models.

classifier h to form an end-to-end system. For instance, in a toxic chemical classification task, given a molecular graph G, it is first mapped to its embedding $z_G = f(G)$ and then classified as $y_G = h(z_G)$. Compared with f, h is typically much simpler (*e.g.*, one fully-connected layer). Note that the data to pre-train f tends to differ from the downstream task but share similar features (*e.g.*, general versus toxic molecules). It is often necessary to *fine-tune* the system. One may opt to perform full-tuning to train both f and h or partial-tuning to only train h but with f fixed [26].

Backdoor attack – Using trojan models as the attack vector, backdoor attacks inject malicious functions into target systems, which are invoked when certain pre-defined conditions ("triggers") are present. Given the increasing use of DNNs in security-critical domains, the adversary is incentivized to forge trojan models and lure users to re-use them. Typically, a trojan model responds to trigger-embedded inputs (*e.g.*, images with specific watermarks) in a highly predictable manner (*e.g.*, misclassified to a particular class) but functions normally otherwise [21, 26, 34]; once it is integrated into a target system [21], the adversary invokes such malicious functions via trigger-embedded inputs during system use.

Threat models – Following the existing work [21, 26, 34, 71], we assume a threat model as shown in Figure 2. Given a pre-trained GNN $f_{\theta_\circ}$ (parameterized by $\theta_\circ$), the adversary forges a trojan GNN f_θ via perturbing its parameters without modifying its architecture (otherwise detectable by checking f's specification). We assume the adversary has access to a dataset $\mathcal{D}$ sampled from the downstream task. Our empirical evaluation shows that often a fairly small amount (*e.g.*, 1%) of the training data from the downstream task suffices (details in § 4). After integrating f_θ with a downstream classifier h to form the end-to-end system, the user performs fine-tuning for the downstream task. To make the attack more practical, we assume the adversary has no knowledge regarding what classifier h is used or how the system is fine-tuned.

3 GTA Attack

At a high level, GTA forges trojan GNNs, which, once integrated into downstream tasks, cause host systems to respond to trigger-embedded graphs in a highly predictable manner.

3.1 Attack overview

For simplicity, we exemplify with the graph classification task to illustrate GTA and discuss its extension to other settings (*e.g.*, transductive learning) in § 3.6.

Given a pre-trained GNN $\theta_\circ$,[2] the adversary aims to forge a trojan model θ so that in the downstream task, θ forces the host system to misclassify all the trigger-embedded graphs to a designated class y_t, while functioning normally on benign graphs. Formally, we define the trigger as a subgraph g_t (including both topological structures and descriptive features), and a *mixing* function $m(\cdot; g_t)$ that blends g_t with a given graph G to generate a trigger-embedded graph $m(G; g_t)$. Therefore, the adversary's objective can be defined as:

$$\begin{cases} h \circ f_\theta(m(G; g_t)) = y_t \\ h \circ f_\theta(G) = h \circ f_{\theta_\circ}(G) \end{cases} \tag{1}$$

where h is the downstream classifier after fine-tuning and G denotes an arbitrary graph in the task. Intuitively, the first objective specifies that all the trigger-embedded graphs are misclassified to the target class (*i.e.*, attack effectiveness), while the second objective ensures that the original and trojan GNNs are indistinguishable in terms of their behaviors on benign graphs (*i.e.*, attack evasiveness).

However, searching for the optimal trigger g_t and trojan model θ in Eq (1) entails non-trivial challenges.

- As the adversary has no access to downstream model h, it is impractical to directly optimize g_t and θ based on Eq (1).
- Due to the mutual dependence of g_t and θ, every time updating g_t requires the expensive re-computation of θ.
- There are combinatorial ways to blend g_t with a given graph G, implying a prohibitive search space.
- Using a universal trigger g_t for all the graphs ignores the characteristics of individual graphs, resulting in suboptimal and easy-to-detect attacks.

To the above challenges, (*i*) instead of associating g_t and θ with final predictions, we optimize them with respect to intermediate representations; (*ii*) we adopt a bi-level optimization formulation, which considers g_t as the hyper-parameters and θ as the model parameters and optimizes them in an interleaving manner; (*iii*) we implement the mixing function $m(G; g_t)$ as an efficient substitution operator, which finds and replaces within G the subgraph g most similar to g_t; and (*iv*) we introduce the concept of adaptive trigger, that is, g_t is specifically optimized for each given graph G.

The overall framework of GTA is illustrated in Figure 3. In the following, we elaborate on each key component.

3.2 Bi-level optimization

Recall that the adversary has access to a dataset $\mathcal{D}$ sampled from the downstream task, which comprises a set of instances (G, y_G) with G being a graph and y_G as its class. We formulate

[2] As GTA does not modify the model architecture, below we use θ to refer to both the model and it parameter configuration.

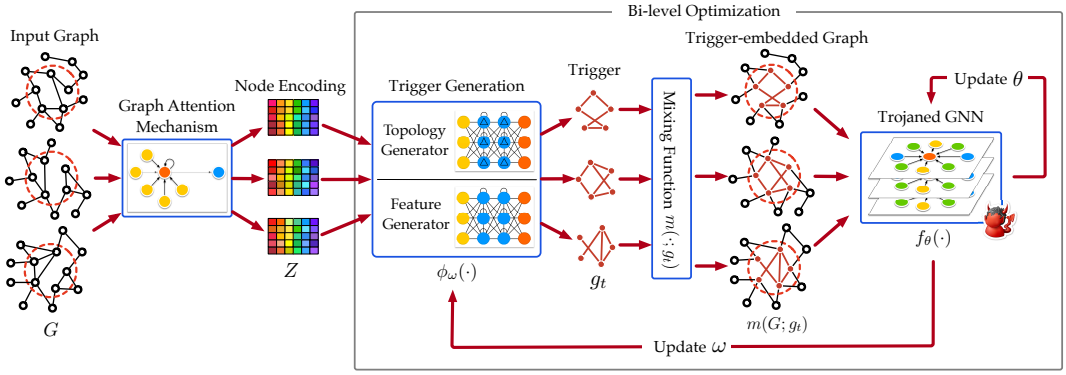


Figure 3: Overall framework of GTA attack.

the bi-level optimization objective [16] with g_t and θ as the upper- and lower-level variables:

$$\begin{aligned} g_t^* &= \arg\min_{g_t} \ell_{\text{atk}}(\theta^*(g_t), g_t) \\ \text{s.t.} \quad \theta^*(g_t) &= \arg\min_{\theta} \ell_{\text{ret}}(\theta, g_t) \end{aligned} \tag{2}$$

where ℓ_{atk} and ℓ_{ret} represent the loss terms respectively quantifying attack effectiveness and accuracy retention, corresponding to the objectives defined in Eq (1).

Without access to downstream classifier h, instead of associating ℓ_{atk} and ℓ_{ret} with final predictions, we define them in terms of latent representations. We partition $\mathcal{D}$ into two parts, $\mathcal{D}[y_t]$ – the graphs in the target class y_t and $\mathcal{D}[\setminus y_t]$ – the ones in the other classes; ℓ_{atk} enforces that f_θ generates similar embeddings for the graphs in $\mathcal{D}[y_t]$ and those in $\mathcal{D}[\setminus y_t]$ once embedded with g_t. Meanwhile, ℓ_{ret} ensures that f_θ and $f_{\theta_\circ}$ produce similar embeddings for the graphs in $\mathcal{D}$. Formally,

$$\ell_{\text{atk}}(\theta, g_t) = \mathbb{E}_{G \in \mathcal{D}[\setminus y_t], G' \in \mathcal{D}[y_t]} \Delta\left(f_\theta(m(G; g_t))), f_\theta(G')\right) \tag{3}$$

$$\ell_{\text{ret}}(\theta, g_t) = \mathbb{E}_{G \in \mathcal{D}} \Delta\left(f_\theta(G), f_{\theta_\circ}(G)\right) \tag{4}$$

where $\Delta(\cdot,\cdot)$ measures the embedding dissimilarity, which is instantiated as L_2 distance in our current implementation.

However, exactly solving Eq (2) is expensive. Due to the bi-level formulation, it requires re-computing θ (*i.e.*, re-training f over $\mathcal{D}$) whenever g_t is updated. Instead, we propose an approximate solution that iteratively optimizes g_t and θ by alternating between gradient descent on ℓ_{atk} and ℓ_{ret}.

Specifically, at the i-th iteration, given the current trigger $g_t^{(i-1)}$ and model $\theta^{(i-1)}$, we first compute $\theta^{(i)}$ by gradient descent on ℓ_{ret}, with $g_t^{(i-1)}$ fixed. In practice, we may run this step for n_{io} iterations. The parameter n_{io}, *inner-outer optimization ratio*, essentially balances the optimization of ℓ_{atk} and ℓ_{ret}. We then obtain $g_t^{(i)}$ by minimizing ℓ_{atk} after a single look-ahead step of gradient descent with respect to $\theta^{(i)}$. Formally, the gradient with respect to g_t is approximated by:

$$\begin{aligned} &\nabla_{g_t} \ell_{\text{atk}}\left(\theta^*(g_t^{(i-1)}), g_t^{(i-1)}\right) \\ \approx &\nabla_{g_t} \ell_{\text{atk}}\left(\theta^{(i)} - \xi \nabla_\theta \ell_{\text{ret}}\left(\theta^{(i)}, g_t^{(i-1)}\right), g_t^{(i-1)}\right) \end{aligned} \tag{5}$$

where ξ is the learning rate of the look-ahead step.

Intuitively, while it is expensive to optimize $\ell_{\text{atk}}(\theta^*(g_t), g_t)$ with respect to g_t, we use a single-step unrolled model [16] as a surrogate of $\theta^*(g_t)$ (details in § A.1).

3.3 Mixing function

The mixing function $m(G; g_t)$ fulfills two purposes: (*i*) for a given trigger g_t, it identifies the optimal to-be-replaced subgraph g within a given graph G; and (*ii*) it performs the substitution of g with g_t. Apparently, there are combinatorial ways to define $m(G; g_t)$, resulting in a prohibitive search space.

To address this challenge, we restrict the mixing function to an efficient substitution operator; that is, $m(G; g_t)$ replaces a subgraph g in G with g_t. To maximize the attack evasiveness, it is desirable to use a subgraph similar to g_t. We thus specify the constraints that (*i*) g and g_t are of the same size (*i.e.*, the same number of nodes) and (*ii*) they have the minimum graph edit distance (*i.e.*, edge addition or deletion).

It is known that finding in a given graph G a subgraph g identical to g_t (subgraph isomorphism) is NP-hard. We adapt a backtracking-based algorithm VF2 [41] to our setting. Intuitively, VF2 recursively extends a partial match by mapping the next node in g_t to a node in G if feasible, and backtracks otherwise. As we search for the most similar subgraph, we maintain the current highest similarity and terminate a partial match early if it exceeds this threshold. The detailed implementation is deferred to § A.2.

3.4 Trigger generation

In the formulation of Eq (2), we assume a universal trigger for all the graphs. Despite its simplicity for implementation, fixing the trigger entails much room for optimization: (*i*) it ignores the characteristics of individual graphs and results in less effective attacks; (*ii*) it becomes a pattern shared by trigger-embedded graphs and makes them easily detectable. We thus postulate whether it is possible to generate triggers tailored to individual graphs to maximize the attack effectiveness and evasiveness [35, 52].

We design an adaptive trigger generation function $\phi_\omega(\cdot)$, which proposes a trigger g_t tailored to a given subgraph g

within G. At a high level, $\phi_\omega(\cdot)$ comprises two key operations: (*i*) it first maps each node i in g to its encoding z_i, which encodes both g's node features and topological structures; (*ii*) it applies two generator functions structured by neural networks, the first mapping g's node encodings to g_t's topological structures and the second mapping g's node encodings to g_t's node features. Next, we elaborate on the design of $\phi_\omega(\cdot)$.

How to encode g's features and context? To encode g's topological structures and node features as well as its context within G, we resort to the recent advances of graph attention mechanisms [59]. Intuitively, for a given pair of nodes i, j, we compute an attention coefficient α_{ij} specifying j's importance with respect to i, based on their node features and topological relationship; we then generate i's encoding as the aggregation of its neighboring encodings (weighted by their corresponding attention coefficients) after applying a non-linearity transformation. We train the attention network (details in Table 10) using $\mathcal{D}$. Below we denote by $z_i \in \mathbb{R}^d$ the encoding of node i (d is the encoding dimensionality).

How to map g's encoding to g_t? Recall that g_t comprises two parts, its topological structures and node features.

Given two nodes $i, j \in g$ with their encodings z_i and z_j , we define their corresponding connectivity $\tilde{A}_{ij}$ in g_t using their parameterized cosine similarity:

$$\tilde{A}_{ij} = \mathbb{1}_{z_i^\top W_c^\top W_c z_j \geq \|W_c z_i\| \|W_c z_j\|/2} \tag{6}$$

where $W_c \in \mathbb{R}^{d\times d}$ is learnable and $\mathbb{1}_p$ is an indicator function returning 1 if p is true and 0 otherwise. Intuitively, i and j are connected in g_t if their similarity score exceeds 0.5.

Meanwhile, for node $i \in g$, we define its feature $\tilde{X}_i$ in g_t as

$$\tilde{X}_i = \sigma(W_f z_i + b_f) \tag{7}$$

where $W_f \in \mathbb{R}^{d\times d}$ and $b_f \in \mathbb{R}^d$ are both learnable, and $\sigma(\cdot)$ is a non-linear activation function.

In the following, we refer to W_c, W_f, and b_f collectively as ω, and the mapping from g's encoding to $\{\tilde{X}_i\}$ ($i \in g$) and $\{\tilde{A}_{ij}\}$ ($i, j \in g$) as the trigger generation function $\phi_\omega(g)$.

How to resolve the dependence of g and g_t? Astute readers may point out that the mixing function $g = m(G; g_t)$ and the trigger generation function $g_t = \phi_\omega(g)$ are mutually dependent: the generation of g_t relies on g, while the selection of g depends on g_t. To resolve this "chicken-and-egg" problem, we update g and g_t in an interleaving manner.

Specifically, initialized with a randomly selected g, at the i-th iteration, we first update the trigger $g_t^{(i)}$ based on $g^{(i-1)}$ from the $(i-1)$-th iteration and then update the selected subgraph $g^{(i)}$ based on $g_t^{(i)}$. In practice, we limit the number of iterations by a threshold n_{iter} (*cf.* Table 10).

3.5 Implementation and optimization

Putting everything together, Algorithm 1 sketches the flow of GTA attack. At its core, it alternates between updating the model θ, the trigger generation function $\phi_\omega(\cdot)$, and the selected subgraph g for each $G \in \mathcal{D}[\setminus y_t]$ (line 4 to 6). Below we present a suite of optimization to improve the attack.

Algorithm 1: GTA (inductive) attack

Input: $\theta_\circ$ - pre-trained GNN; $\mathcal{D}$ - data from downstream task; y_t - target class;
Output: θ - trojan GNN; ω - parameters of trigger generation function

```
// initialization
1 randomly initialize ω;
2 foreach G ∈ D[\y_t] do randomly sample g ∼ G;
// bi-level optimization
3 while not converged yet do
      // updating trojan GNN
4     update θ by descent on ∇_θ ℓ_ret(θ, g_t) (cf. Eq (4));
      // updating trigger generation function
5     update ω by descent on ∇_ω ℓ_atk(θ − ξ∇_θ ℓ_ret(θ, g_t), g_t) (cf. Eq (5));
      // updating subgraph selection
6     for G ∈ D[\y_t] do update g with m(G; φ_ω(g));
7 return (θ, ω);
```

Periodical reset – Recall that we update the model with gradient descent on ℓ_{ret}. As the number of update steps increases, this estimate may deviate significantly from the true model trained on $\mathcal{D}$, which negatively impacts the attack effectiveness. To address this, periodically (*e.g.*, every 20 iterations), we replace the estimate with the true model $\theta^*(g_t)$ thoroughly trained based on the current trigger g_t.

Subgraph stabilization – It is observed in our empirical evaluation that stabilizing the selected subgraph g for each $G \in \mathcal{D}[\setminus y_t]$ by running the subgraph update step (line 6) for multiple iterations (*e.g.*, 5 times), with the trigger generation function fixed, often leads to faster convergence.

Model restoration – Once trojan GNN f_θ is trained, the adversary may opt to restore classifier $h_\circ$ (not the downstream classifier h) with respect to the pre-training task. Due to the backdoor injection, $h_\circ$ may not match f_θ. The adversary may fine-tune $h_\circ$ using the training data from the pre-training task. This step makes the accuracy of the released model $h_\circ \circ f_\theta$ match its claims, thereby passing model inspection [71].

3.6 Extension to transductive learning

We now discuss the extension of GTA to a transductive setting: given a graph G and a set of labeled nodes, the goal is to infer the classes of the remaining unlabeled nodes $\mathcal{V}_U$ [78].

We assume the following setting. The adversary has access to G as well as the classifier. For simplicity, we denote by $f_\theta(v; G)$ the complete system that classifies a given node v within G. Further, given an arbitrary subgraph g in G, by substituting g with the trigger g_t, the adversary aims to force the unlabeled nodes within K hops to g to be misclassified to the target class y_t, where K is the number GNN layers. Recall that for neighborhood aggregation-based GNNs, a node exerts its influence to other nodes at most K hops away; this goal upper-bounds the attack effectiveness.

Dataset	# Graphs ($\lvert\mathcal{G}\rvert$)	Avg. # Nodes ($\lvert\mathcal{V}\rvert$)	Avg. # Edges ($\lvert\mathcal{E}\rvert$)	# Classes ($\lvert\mathcal{Y}\rvert$)	# Graphs [Class]	Target Class y_t
Fingerprint	1661	8.15	6.81	4	538 [0], 517 [1], 109 [2], 497 [3]	2
WinMal	1361	606.33	745.34	2	546 [0], 815 [1]	0
AIDS	2000	15.69	16.20	2	400 [0], 1600 [1]	0
Toxicant	10315	18.67	19.20	2	8982 [0], 1333 [1]	1
AndroZoo	211	5736.5	25234.9	2	109 [0], 102 [1]	1
Bitcoin	1	5664	19274	2	1556 [0], 4108 [1]	0
Facebook	1	12539	108742	4	4731 [0], 1255 [1], 2606 [2], 3947 [3]	1

Table 1. Dataset statistics: # Graphs - number of graphs in the dataset; Avg. # Nodes - average number of nodes per graph; Avg. # Edges - average number of edges per graph; # Classes - number of classes; # Graph [Class] - number of graphs in each [class]; Target Class - target class designated by the adversary.

Dataset	Setting	GNN	Accuracy
Fingerprint	Inductive (Fingerprint→Fingerprint)	GAT	82.9%
WinMal	Inductive (WinMal→WinMal)	GRAPHSAGE	86.5%
AIDS	Inductive (Toxicant→AIDS)	GCN	93.9%
Toxicant	Inductive (AIDS→Toxicant)	GCN	95.4%
AIDS	Inductive (ChEMBL→AIDS)	GCN	90.4%
Toxicant	Inductive (ChEMBL→Toxicant)	GCN	94.1%
Bitcoin	Transductive	GAT	96.3%
Facebook	Transductive	GRAPHSAGE	83.8%
AndroZoo	Inductive (Topology Only)	GCN	95.3%
	Inductive (Topology + Feature)	GCN	98.1%

Table 2. Accuracy of clean GNN models ($\mathcal{T}_{\text{ptr}} \rightarrow \mathcal{T}_{\text{dst}}$ indicates the transfer from pre-training domain $\mathcal{T}_{\text{ptr}}$ to downstream domain $\mathcal{T}_{\text{dst}}$).

We re-define the loss functions in Eq (3) and (4) as:

$$\ell_{\text{atk}}(\theta, g_t) = \mathbb{E}_{g\sim G}\mathbb{E}_{v\in\mathcal{N}_k(g)}\ell(f_\theta(v; G\ominus g\oplus g_t), y_t) \quad (8)$$

$$\ell_{\text{ret}}(\theta, g_t) = \mathbb{E}_{g\sim G}\mathbb{E}_{v\in\mathcal{V}_U\setminus\mathcal{N}_k(g)}\ell(f_\theta(v; G\ominus g\oplus g_t), f_{\theta_\circ}(v; G)) \quad (9)$$

where $\mathcal{N}_k(g)$ is the set of nodes within K hops of g, $G\ominus g\oplus g_t$ is G after substituting g with g_t, and $\ell(\cdot,\cdot)$ is a proper loss function (*e.g.*, cross entropy). Also, given that g is selected by the adversary, the mixing function is not necessary. The complete attack is sketched in Algorithm 3.

4 Attack Evaluation

Next, we conduct an empirical study of GTA to answer the following key questions:

Q1 – How effective/evasive is GTA in inductive tasks?
Q2 – How effective is it on pre-trained, off-the-shelf GNNs?
Q3 – How effective/evasive is it in transductive tasks?
Q4 – Is GTA agnostic to downstream models?

Experimental settings

Datasets – We primarily use 7 datasets drawn from security-sensitive domains. (*i*) Fingerprint [40] – graph representations of fingerprint shapes from the NIST-4 database [65]; (*ii*) WinMal [46] – Windows PE call graphs of malware and goodware; (*iii*) AIDS [47] and (*iv*) Toxicant [56] – molecular structure graphs of active and inactive compounds; (*v*) AndroZoo - call graphs of benign and malicious APKs collected from AndroZoo [1]; (*vi*) Bitcoin [14] – an anonymized Bitcoin transaction network with each node (transaction) labeled as legitimate or illicit; and (*vii*) Facebook [48] – a page-page relationship network with each node (Facebook page) annotated with the page properties (*e.g.*, place, organization, product). The dataset statistics are summarized in Table 1. Among them, we use the datasets (*i-v*) for the inductive setting and the rest (*vi-vii*) for the transductive setting.

Models – In our evaluation, we use 3 state-of-the-art GNN models: GCN [28], GRAPHSAGE [23, 24], and GAT [59]. Using GNNs of distinct network architectures (*i.e.*, graph convolution, general aggregation function, versus graph attention), we factor out the influence of the characteristics of individual models. The performance of systems built upon clean GNN models is summarized in Table 2.

Baselines – To our best knowledge, GTA is the first backdoor attack on GNNs. We thus mainly compare GTA with its variants as baselines: BL^{I}, which fixes the trigger as a complete subgraph and optimizes a feature vector shared by all its nodes, and BL^{II}, which optimizes the trigger's connectivity and the feature vector of each of its nodes. Both BL^{I} and BL^{II} assume a universal trigger for all the graphs, while GTA optimizes the trigger's topological connectivity and node features with respect to each graph. Intuitively, BL^{I}, BL^{II}, and GTA represent different levels of trigger adaptiveness.

In each set of experiments, we apply the same setting across all the attacks, with the default parameter setting summarized in Table 10. In particular, in each dataset, we assume the class with the smallest number of instances to be the target class y_t designated by the adversary (*cf.* Table 1), to minimize the impact of unbalanced data distributions.

Metrics – To evaluate attack effectiveness, we use two metrics: (*i*) *attack success rate* (*ASR*), which measures the likelihood that the system classifies trigger-embedded inputs to the target class y_t designated by the adversary:

$$\text{Attack Success Rate } (ASR) = \frac{\text{\# successful trials}}{\text{\# total trials}} \quad (10)$$

and (*ii*) *average misclassification confidence* (*AMC*), which is the average confidence score assigned to class y_t by the system with respect to successful attacks. Intuitively, higher *ASR* and *AMC* indicate more effective attacks.

To evaluate the attack evasiveness, we use four metrics: (*i*) *clean accuracy drop* (*CAD*), which measures the difference of classification accuracy of two systems built upon the original GNN and its trojan counterpart with respect to clean graphs; (*ii*) *average degree difference* (*ADD*), (*iii*) *average eccentricity change* (*AEC*), and (*iv*) *algebraic connectivity change* (*ACC*),

Setting	Available Data ($\lvert\mathcal{D}\rvert/\lvert\mathcal{T}\rvert$)	Attack Effectiveness (ASR | AMC)			Attack Evasiveness (CAD | ADD | AEC | ACC)		
		BL^{I}	BL^{II}	GTA	BL^{I}	BL^{II}	GTA
Fingerprint ↻	\	84.4% .862	87.2% .909	100% .997	1.9% 2.8×10^{-3} 1.6% 8.4%	1.6% 5.6×10^{-4} 0.9% 2.6%	0.9% 4.3×10^{-4} 0.9% 1.7%
WinMal ↻		87.2% .780	94.4% .894	100% .973	1.8% 5.6×10^{-4} 0.1% 0.8%	1.2% 6.1×10^{-6} 0.0% 0.0%	0.0% 2.1×10^{-5} 0.0% 0.0%
Toxicant →AIDS	0.2%	64.1% .818	70.2% .903	91.4% .954	2.3%	2.5%	2.1%
	1%	89.4% .844	95.5% .927	98.0% .996	1.7% 1.6×10^{-2} 2.3% 5.3%	1.3% 9.3×10^{-3} 2.0% 4.0%	1.4% 7.6×10^{-3} 1.7% 3.3%
	5%	91.3% .918	97.2% .947	100% .998	0.4%	0.6%	0.2%
AIDS →Toxicant	0.2%	73.5% .747	77.8% .775	94.3% .923	1.3%	0.6%	1.0%
	1%	80.2% .903	85.5% .927	99.8% .991	0.6% 1.4×10^{-2} 2.4% 5.6%	0.0% 5.5×10^{-3} 1.6% 1.2%	0.4% 6.9×10^{-3} 1.2% 1.1%
	5%	84.6% .935	86.1% .976	100% .998	0.1%	0.0%	0.0%

Table 3. Attack effectiveness and evasiveness of GTA in inductive tasks ($\mathcal{T}_{\text{ptr}} \to \mathcal{T}_{\text{dst}}$ indicates transfer from pre-training task $\mathcal{T}_{\text{ptr}}$ to downstream task $\mathcal{T}_{\text{dst}}$).

which respectively measure the difference of average degrees, eccentricity, and algebraic connectivity of clean graphs and their trigger-embedded counterparts.

Q_1: Is GTA effective in inductive tasks?

This set of experiments evaluate GTA under the inductive setting, in which a pre-trained GNN is used in a downstream graph classification task. Based on the relationship between pre-training and downstream tasks, we consider two scenarios.

(*i*) Non-transfer – In the case that the two tasks share the same dataset, we partition the overall dataset $\mathcal{T}$ into 40% and 60% for the pre-training and downstream tasks respectively. We assume the adversary has access to 1% of $\mathcal{T}$ (as $\mathcal{D}$) to forge trojan models. In the evaluation, we randomly sample 25% from the downstream dataset to construct trigger-embedded graphs and the rest as clean inputs.

(*ii*) Transfer – In the case that the two tasks use different datasets, in the pre-training task, we use the whole dataset for GNN pre-training; in the downstream task, we randomly partition the dataset $\mathcal{T}$ into 40% and 60% for system fine-tuning and testing respectively. By default, we assume the adversary has access to 1% of $\mathcal{T}$. Similar to the non-transfer case, we sample 25% from the testing set of $\mathcal{T}$ to build trigger-embedded graphs and the rest as clean inputs.

In both cases, we assume the adversary has no knowledge regarding downstream models or fine-tuning strategies. By default, we use a fully-connected layer plus a softmax layer as the downstream classifier and apply full-tuning over both the GNN and the classifier.

Attack efficacy – Table 3 summarizes the performance of different variants of GTA in inductive tasks. Overall, in both non-transfer and transfer settings, all the attacks achieve high attack effectiveness (each with an attack success rate over 80.2% and misclassification confidence over 0.78), effectively retain the accuracy of pre-trained GNNs (with accuracy drop below 1.9%), and incur little impact on the statistics of input graphs (with average degree difference below 0.016), which highlights the practicality of backdoor attacks against GNN models. The attacks are ranked as GTA > BL^{II} > BL^{I} in terms of *ASR*. This may be explained by that the trigger adaptiveness exploits the characteristics of individual graphs, leading to more effective attacks. Note that in the transfer cases, BL^{II} attains slightly higher evasiveness (accuracy retention) than GTA. This is perhaps because given its higher flexibility, to retain the accuracy over clean inputs, GTA requires more data from the downstream task to constrain its optimization. To validate this hypothesis, we increase the amount of $\mathcal{T}$ accessible by the adversary to 5%. Observe that under this setting GTA attains the highest accuracy retention.

Trigger size n_{trigger} – We now evaluate the impact of trigger size n_{trigger} on GTA. Intuitively, n_{trigger} specifies the number of nodes in the trigger subgraph. Figure 4 measures the effectiveness (*ASR*) and evasiveness (*CAD*) of different attacks as n_{trigger} varies from 2 to 6. Observe that the effectiveness of all the attacks monotonically increases with n_{trigger}, which is especially evident for BL^{I} and BL^{II}. Intuitively, with larger triggers, the trojan GNNs are able to better differentiate trigger-embedded and clean graphs. In comparison, as GTA enjoys the flexibility of adapting triggers to individual graphs, its effectiveness is less sensitive to n_{trigger}. Meanwhile, the attack evasiveness of all the attacks marginally decreases as n_{trigger} grows (less than 3.6%). This may be explained by that as larger triggers represent more significant graph patterns, the trojan GNNs need to dedicate more network capacity to recognize such patterns, which negatively interferes with the primary task of classifying clean graphs.

Inner-outer optimization ratio n_{io} – Recall that in the bi-level optimization framework (*cf.* Eq (2)), the inner-outer optimization ratio n_{io} specifies the number of iterations of optimizing ℓ_{ret} per iteration of optimizing ℓ_{atk}, which balances the attack effectiveness and evasiveness: by increasing n_{io}, one emphasizes more on minimizing the difference of original and trojan GNNs on clean inputs. Figure 5 illustrates the performance of GTA as a function of n_{io} in the inductive tasks. Observe that across all the cases both the *ASR* and *CAD* measures decrease with n_{io}, highlighting their inherent trade-off. Also note that among the three attacks, GTA is the least sensitive to n_{io}. This may be explained by that introducing trigger adaptiveness admits a larger optimization space to improve both effectiveness and evasiveness.

Q_2: Is GTA effective on off-the-shelf GNNs?

Besides models trained from scratch, we further consider pre-trained GNNs "in the wild". We use a GCN model[3] that is pre-

[3] https://github.com/snap-stanford/pre-train-gnns/

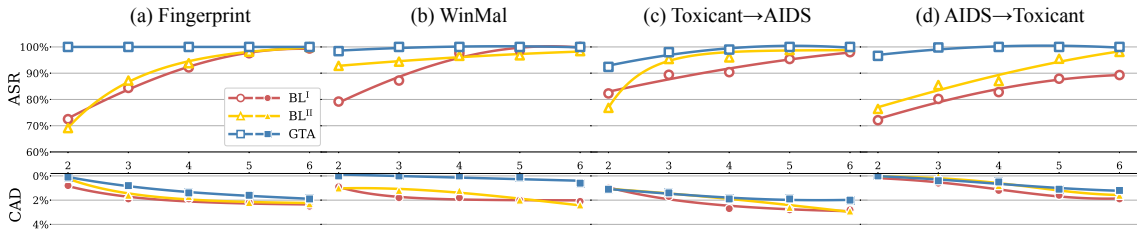


Figure 4: Impact of trigger size n_{trigger} on the attack effectiveness and evasiveness of GTA in inductive tasks.

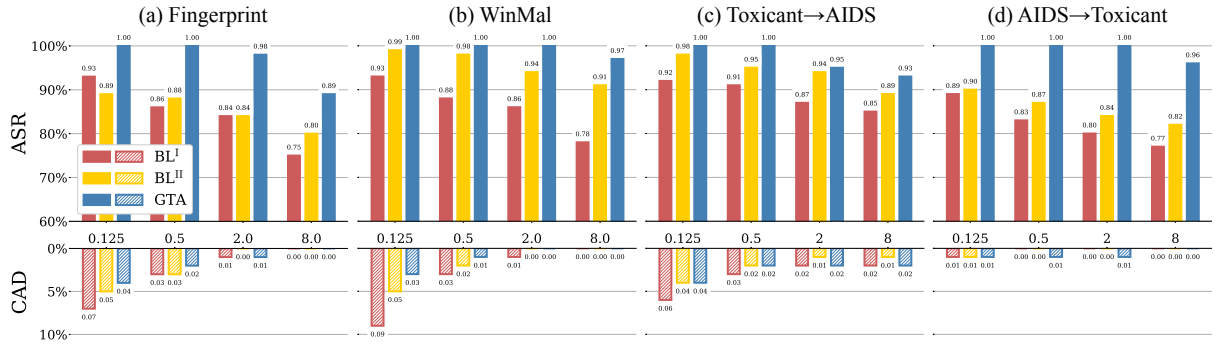


Figure 5: Impact of inner-outer optimization ratio n_{io} on the trade-off of attack effectiveness and evasiveness in inductive tasks.

trained with graph-level multi-task supervised training [25] on the ChEMBL dataset [38], containing 456K molecules with 1,310 kinds of diverse biochemical assays. We transfer this model to the tasks of classifying the AIDS and Toxicant datasets. The default setting is identical to the transfer case.

Attack efficacy – Table 4 summarizes the attack efficacy of GTA on the pre-trained GNN under varying settings of the available data ($|\mathcal{D}|/|\mathcal{T}|$). We have the observations below.

First, across all the cases, the three attacks are ranked as GTA > BL^{II} > BL^{I} in terms of their effectiveness, highlighting the advantage of using flexible trigger definitions.

Second, the effectiveness of GTA increases as more data from the downstream task becomes available. For instance, the *ASR* of BL^{I} grows about 30% as $|\mathcal{D}|/|\mathcal{T}|$ increases from 0.2 to 5% on AIDS. In comparison, GTA is fairly insensitive to the available data. For instance, with $|\mathcal{D}|/|\mathcal{T}| = 0.2\%$, it attains over 92.5% *ASR* on Toxicant.

Third, by comparing Table 3 and 4, it is observed that GTA appears slightly more effective on the off-the-shelf GNN. For instance, with $|\mathcal{D}|/|\mathcal{T}| = 5\%$, BL^{II} attains 86.1% and 94.1% *ASR* on the trained-from-scratch and off-the-shelf GNN models respectively on AIDS. This is perhaps explained by that the models pre-trained under the multi-task supervised setting tend to have superior transferability to downstream tasks [25], which translates into more effective backdoor attacks and less reliance on available data.

Trigger size n_{trigger} – We then evaluate the impact of trigger size n_{trigger} on GTA. Figure 6 shows the effectiveness (*ASR*) and evasiveness (*CAD*) of GTA as a function of n_{trigger}. It is observed that similar to Figure 4, the effectiveness of all the attacks monotonically increases with n_{trigger} and meanwhile their evasiveness marginally drops (less than 3.6%). It seems that among the three attacks GTA achieves the best balance between the two objectives, which is perhaps attributed to the flexibility bestowed by the trigger adaptiveness.

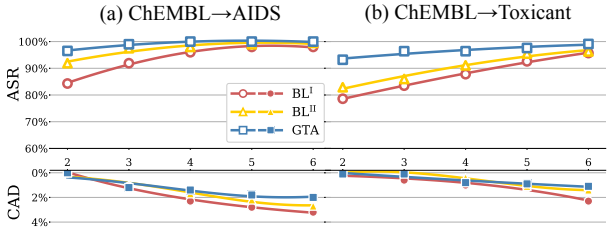


Figure 6: Impact of trigger size n_{trigger} on the attack effectiveness and evasiveness of GTA against off-the-shelf models.

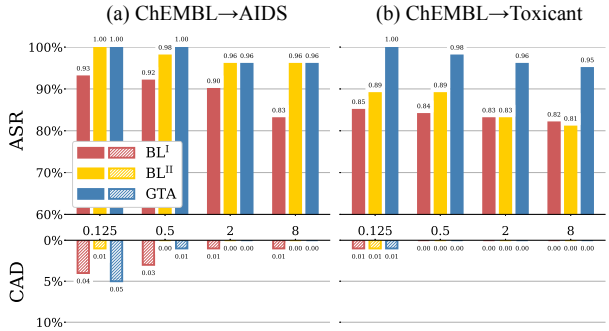


Figure 7: Impact of inner-outer optimization ratio n_{io} on the attack effectiveness and evasiveness of GTA against off-the-shelf models.

Inner-outer optimization ratio n_{io} – Figure 7 illustrates the performance of GTA as the inner-outer optimization ratio n_{io} varies from 0.125 to 8, which shows trends highly similar to the transfer cases in Figure 5: of all the attacks, their effectiveness and evasiveness respectively show positive and negative correlation with n_{io}, while GTA is the least sensitive to n_{io}. Given the similar observations on both trained-from-scratch and off-the-shelf GNNs, it is expected that with proper configuration, GTA is applicable to a range of settings.

Setting	Available Data ($\lvert\mathcal{D}\rvert/\lvert\mathcal{T}\rvert$)	Attack Effectiveness (ASR | AMC) BL$^{\text{I}}$	BL$^{\text{II}}$	GTA	Attack Evasiveness (CAD | ADD | AEC | ACC) BL$^{\text{I}}$	BL$^{\text{II}}$	GTA
ChEMBL →AIDS	0.2%	68.2% .805	77.3% .796	94.4% .937	1.3%	2.2%	1.5%
	1%	92.0% .976	97.5% .994	99.0% .994	1.1% 1.6×10^{-2} 2.3% 6.5%	1.0% 9.2×10^{-3} 2.1% 5.9%	1.2% 7.6×10^{-3} 1.5% 2.5%
	5%	98.1% .992	100% .987	100% .995	0.4%	0.7%	0.3%
ChEMBL → Toxicant	0.2%	78.0% .847	78.8% .876	92.5% .915	0.7%	0.3%	0.4%
	1%	83.5% .929	86.0% .940	96.4% .971	0.6% 1.4×10^{-2} 2.4% 8.1%	0.0% 8.5×10^{-3} 1.4% 1.7%	0.1% 7.0×10^{-3} 1.4% 1.1%
	5%	92.7% .956	94.1% .983	99.2% .995	0.3%	0.0%	0.0%

Table 4. Performance of GTA against pre-trained, off-the-shelf GNN models.

Dataset	Effectiveness (ASR% | AMC) BL$^{\text{I}}$	BL$^{\text{II}}$	GTA	Evasiveness (CAD%) BL$^{\text{I}}$	BL$^{\text{II}}$	GTA
Bitcoin	52.1 .894	68.6 .871	89.7 .926	0.9	1.2	0.9
Facebook	42.6 .903	59.6 .917	69.1 .958	4.0	2.9	2.4

Table 5. Performance of GTA in transductive tasks.

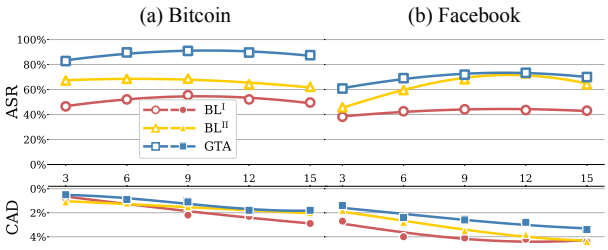


Figure 8: Impact of trigger size n_{trigger} on the attack effectiveness and evasiveness of GTA in transductive tasks.

Q_3: Is GTA effective in transductive tasks?

We now evaluate GTA under the transductive setting, in which given a graph and a set of labeled nodes, the system classifies the remaining unlabeled nodes. Specifically, given a subgraph g in G (designated by the adversary), by replacing g with the trigger g_t, the adversary aims to force all the unlabeled nodes within K hops of g_t (including g_t) to be classified to target class y_t, where K is the number of layers of the GNN.

In each task, we randomly partition G's nodes into 20% as the labeled set $\mathcal{V}_{\text{L}}$ and 80% as the unlabeled set $\mathcal{V}_{\text{U}}$. We then randomly sample 100 subgraphs from G as the target subgraphs $\{g\}$. Similar to the inductive attacks, we measure the attack effectiveness and evasiveness using *ASR* (*AMC*) and *CAD* respectively. In particular, *ASR* (*AMC*) is measured over the unlabeled nodes within K hops of g, while *CAD* is measured over all the other unlabeled nodes.

Attack efficacy – Table 5 summarizes the attack performance of GTA. Similar to the inductive case (*cf.* Table 3), GTA outperforms the rest by a larger margin in the transductive tasks. For instance, on Bitcoin, GTA attains 37.6% and 21.1% higher *ASR* than BL$^{\text{I}}$ and BL$^{\text{II}}$ respectively. This is explained as follows. Compared with the inductive tasks, the graphs in the transductive tasks tend to be much larger (*e.g.*, thousands versus dozens of nodes) and demonstrate more complicated topological structures; being able to adapt trigger patterns to local topological structures significantly boosts the attack effectiveness. Further, between the two datasets, the attacks attain higher *ASR* on Bitcoin, which may be attributed to that all the node features in Facebook are binary-valued, negatively impacting the effectiveness of feature perturbation.

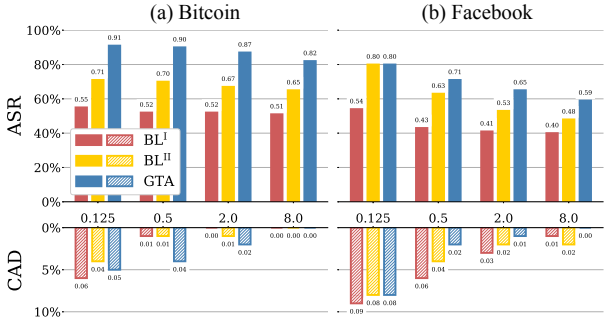


Figure 9: Impact of inner-outer optimization ratio n_{io} on the attack effectiveness and evasiveness of GTA in transductive tasks.

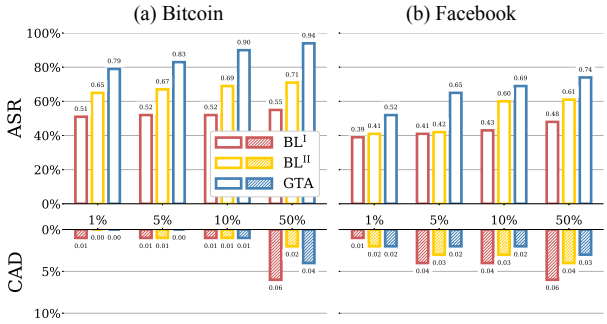
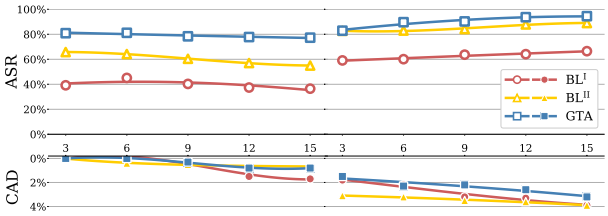


Figure 10: Impact of feature mask size n_{mask} on the attack effectiveness and evasiveness of GTA in transductive tasks.

Trigger size n_{trigger} – Figure 8 shows the impact of trigger size n_{trigger}. Observe that as n_{trigger} varies from 3 to 15, the *ASR* of all the attacks first increases and then slightly drops. We have a possible explanation as follow. The "influence" of trigger g_t on its neighborhood naturally grows with n_{trigger}; meanwhile, the number of unlabeled nodes $\mathcal{N}_K(g_t)$ within g_t's vicinity also increases super-linearly with n_{trigger}. Once the increase of $\mathcal{N}_K(g_t)$ outweighs g_t's influence, the attack effectiveness tends to decrease. Interestingly, n_{trigger} seems have limited impact on GTA's *CAD*, which may be attributed to that g_t's influence is bounded by K hops.

Inner-outer optimization ratio n_{io} – Figure 9 shows the efficacy of GTA as a function of the inner-outer optimization ratio n_{io}. The observations are similar to the inductive case (*cf.* Figure 5): of all the attacks, their effectiveness and evasiveness respectively show positive and negative correlation with n_{io}, while GTA is the least sensitive to n_{io}.

Feature mask size n_{mask} – Recall that one may optimize GTA by limiting the number of perturbable features at each node of the to-be-replaced subgraph (§ 3.5). We now evaluate the impact of feature mask size n_{mask}, which specifies



(a) (b)

Figure 11: Interactions of trigger size n_{trigger} and feature mask size n_{mask} on Bitcoin: (a) $n_{\text{mask}} = 1\%$; (b) $n_{\text{mask}} = 50\%$.

Classifier	Accuracy (%)	Effectiveness (ASR%)			Evasiveness (CAD%)		
		BL$^{\text{I}}$	BL$^{\text{II}}$	GTA	BL$^{\text{I}}$	BL$^{\text{II}}$	GTA
NB	95.4	87.7	92.4	99.5	1.5	0.9	0.7
RF	97.4	85.8	88.0	90.1	0.9	0.9	0.6
GB	97.4	82.7	89.3	94.0	0.6	0.6	0.6

Table 6. Performance of GTA with respect to different downstream classifiers: NB - Naïve Bayes; RF - Random Forest; GB - Gradient Boosting.

the percentage of perturbable features, with results shown in Figure 10. Observe that the attack effectiveness shows strong correlation with n_{mask}. As n_{mask} varies from 1% to 50%, the *ASR* of GTA increases by 15% on Bitcoin. Intuitively, larger perturbation magnitude leads to more effective attacks. Meanwhile, n_{mask} negatively impacts the attack evasiveness, which is especially evident on Facebook. This can be explained by: (*i*) unlike other parameters (*e.g.*, n_{trigger}), as it affects the feature extraction of all the nodes, n_{mask} has a "global" impact on the GNN behaviors; and (*ii*) as all the features of Facebook are binary-valued, n_{mask} tends to have a larger influence.

We are also interested in understanding the interplay between n_{trigger} and n_{mask}, which bound triggers in terms of topology and feature perturbation respectively. Figure 11 compares the attack efficacy (as a function of n_{trigger}) under $n_{\text{mask}} = 1\%$ and 50%. When the number of perturbable features is small ($n_{\text{mask}} = 1\%$), increasing the trigger size may negatively impact *ASR*, due to the super-linear increase of neighboring size; when $n_{\text{mask}} = 50\%$, increasing n_{trigger} improves the attack effectiveness, due to the mutual "reinforcement" between feature and topology perturbation; yet, larger n_{mask} also has more significant influence on *CAD*. Therefore, the setting of n_{trigger} and n_{mask} needs to carefully balance these factors.

Q$_4$: Is GTA agnostic to downstream models?

We now instantiate the downstream classifier with alternative models (with the GNN fixed as GCN), including Naïve Bayes (NB), Random Forest (RF), and Gradient Boosting (GB). We evaluate the impact of the classifier on different attacks in the transfer case of ChEMBL→Toxicant, with results in Table 6. Observe that the classifier has a limited impact on GTA. For instance, compared with Table 4 ($|\mathcal{D}|/|\mathcal{T}|$=1%), the *ASR* and *CAD* of GTA vary by less than 9.4% and 0.6%, respectively, implying its insensitivity to the classifier.

Possible explanations – Let $\tilde{G}$ denote an arbitrary trigger-embedded graph. Recall that the optimization of Eq (3) essentially shifts $\tilde{G}$ in the feature space by minimizing $\Delta_{f_\theta}(\tilde{G}) = \|f_\theta(\tilde{G}) - \mathbb{E}_{G\sim P_{y_t}} f_\theta(G)\|$ (with respect to classes other than y_t), where P_{y_t} is the data distribution of target class y_t.

Now consider the end-to-end system $h \circ f_\theta$. Apparently, if $\Delta_{h\circ f_\theta}(\tilde{G}) = \|h \circ f_\theta(\tilde{G}) - \mathbb{E}_{G\sim P_{y_t}} h \circ f_\theta(G)\|$ is minimized (with respect to classes other than y_t), it is likely that $\tilde{G}$ is classified as y_t. One sufficient condition is that $\Delta_{h\circ f_\theta}$ is linearly correlated with Δ_{f_θ}: $\Delta_{h\circ f_\theta} \propto \Delta_{f_\theta}$. If so, we say that the function represented by downstream model h is pseudo-linear [26].

Yet, compared with GNNs, most downstream classifiers are fairly simple and tend to show strong pseudo-linearity. One may thus suggest mitigating GTA by adopting complex downstream models. However, complex models are difficult to train especially when the training data is limited, which is often the case in transfer learning.

Metric	# GCN Layers		
	1	2	3
ASR/AMC	95.4%/.997	98.0%/.996	99.1%/.998
ACC	92.2%	93.9%	95.2%

Table 7. ASR of GTA and overall accuracy as functions of GNN model complexity (Toxicant → AIDS).

5 Discussion

5.1 Causes of attack vulnerabilities

Today's GNNs are complex artifacts designed to model highly non-linear, non-convex functions over graphs. Recent studies [68] show that with GNNs are expressive enough for powerful graph isomorphism tests [66]. These observations may partially explain why, with careful perturbation, a GNN is able to "memorize" trigger-embedded graphs yet without comprising its generalizability on other benign graphs.

To validate this hypothesis, we empirically assess the impact of model complexity on the attack effectiveness of GTA. We use the transfer case of Toxicant →AIDS in § 4 as a concrete example. We train three distinct GCN models with 1-, 2-, and 3-aggregation layers respectively, representing different levels of model complexity. We measure their clean accuracy and the *ASR* of GTA on such models, with results in Table 7.

Observe that increasing model complexity benefits the attack effectiveness. As the layer number varies from 1 to 3, the *ASR* of GTA grows by about 3.7%. We may thus postulate the existence of the correlation between model complexity and attack effectiveness. Meanwhile, increasing model complexity also improves the system performance, that is, the overall accuracy increases by 3%. Therefore, reducing GNN complexity may not be a viable option for defending against GTA, as it may negatively impact system performance.

5.2 Potential countermeasures

As GTA represents a new class of backdoor attacks, one possibility is to adopt the mitigation in other domains (*e.g.*, im-

ages) to defend against GTA. The existing defenses can be roughly classified into two major categories: identifying suspicious models during model inspection (*e.g.*, [9, 33, 60]), and detecting trigger-embedded inputs at inference time (*e.g.*, [7, 11, 13, 17]). We thus extend NeuralCleanse (NC) [60] and Randomized-Smoothing (RS) [75] as the representative defenses of the two categories, and evaluate their effectiveness against GTA (details of RS deferred to § B.1).

Model inspection – We aim to detect suspicious GNNs and potential backdoors at the model inspection stage [9, 33, 60]. We consider NC [60] as a representative method, upon which we build our defense against GTA. Intuitively, given a DNN, NC searches for potential backdoors in every class. If a class is embedded with a backdoor, the minimum perturbation (L_1-norm) necessary to change all the inputs in this class to the target class is abnormally smaller than other classes.

To apply this defense in our context, we introduce the definition below. Given trigger g_t and to-be-replaced subgraph g, let g_t comprise nodes $v_1, \ldots, v_n$ and g correspondingly comprise $u_1, \ldots, u_n$. The cost of substituting g with g_t is measured by the L_1 distance of their concatenated features:

$$\Delta(g_t, g) = \|X_{v_1} \uplus \ldots \uplus X_{v_n} - X_{u_1} \uplus \ldots \uplus X_{u_n}\|_1 \qquad (11)$$

where X_{v_i} is v_i's feature vector (including both its topological and descriptive features) and $\uplus$ denotes the concatenation operator. Intuitively, this measure accounts for both topology and feature perturbation.

We assume a set of benign graphs $\mathcal{D}$. Let $\mathcal{D}_y$ be the subset of $\mathcal{D}$ in class y and $\mathcal{D}_{\bar{y}}$ as the rest. For each class y, we search for the optimal trigger g_t to change the classification of all the graphs in $\mathcal{D}_{\bar{y}}$ to y. The optimality is defined in terms of the minimum perturbation cost (*MPC*):

$$\min_{g_t} \sum_{G \in \mathcal{D}_{\bar{y}}} \min_{g \subset G} \Delta(g_t, g) \quad \text{s.t.} \quad h \circ f_\theta(G \ominus g \oplus g_t) = y \qquad (12)$$

where $G \ominus g \oplus g_t$ denotes G after substituting g with g_t.

We consider three settings for searching for triggers: (i) the trigger g_t^{I} with topology and features universal for all the graphs in $\mathcal{D}_{\bar{y}}$; (ii) the trigger g_t^{II} with universal topology but features adapted to individual graphs; and (iii) the trigger g_t^{III} with both topology and features adapted to individual graphs.

Results and analysis – We evaluate the above defense in the transfer case of pre-trained, off-the-shelf GNN models (ChEMBL→Toxicant). We sample 100 graphs from each class ('0' and '1') of the Toxicant dataset to form $\mathcal{D}$. For comparison, we also run the search on a benign GNN. All the attacks consider '1' as the target class. Figure 12 visualizes the *MPC* measures with respect to each class under varying settings of GNNs, attacks, and trigger definitions.

We have the following observations. First, even on benign models, the *MPC* measure varies across different classes, due to their inherent distributional heterogeneity. Second, on the same model (each column), the measure decreases as the trigger definition becomes more adaptive as tailoring to individual graphs tends to lead to less perturbation. Third, under the same trigger definition (each row), BL^{I} and BL^{II} show significantly disparate *MPC* distributions across the two classes, while the *MPC* distributions of GTA and benign models seem fairly similar, implying the difficulty of distinguishing GNNs trojaned by GTA based on their *MPC* measures.

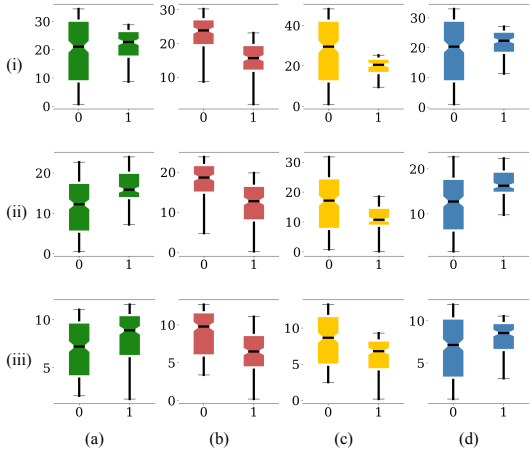


Figure 12: MPC-based backdoor detection (ChEMBL→Toxicant): (i)-(iii) triggers with universal topology and features, universal topology and adaptive features, and adaptive topology and features; (a) benign GNN; (b)-(d) trojan GNNs by BL^{I}, BL^{II}, and GTA.

Trigger Definition	*p*-Value of GNN under Inspection			
	Benign	BL^{I}	BL^{II}	GTA
g_t^{I}	2.6×10^{-9}	1.0×10^{-0}	1.8×10^{-1}	5.4×10^{-7}
g_t^{II}	5.8×10^{-11}	1.0×10^{-0}	2.6×10^{-1}	1.3×10^{-13}
g_t^{III}	1.9×10^{-5}	1.7×10^{-1}	8.4×10^{-2}	9.3×10^{-4}

Table 8. Kolmogorov-Smirnov test of the *MPC* measures of benign and trojan GNNs (ChEMBL → Toxicant).

To validate the observations, on each model, we apply the one-tailed Kolmogorov-Smirnov test [45] between the *MPC* distributions of the two classes, with the null hypothesis being that the *MPC* of the target class is significantly lower than the other class. Table 8 summarizes the results. Observe that regardless of the trigger definition, BL^{I} and BL^{II} show large *p*-values (≥ 0.08), thereby lacking support to reject the null hypothesis; meanwhile, the benign GNN and GTA demonstrate much smaller *p*-values (< 0.001), indicating strong evidence to reject the null hypothesis (*i.e.*, the *MPC* of the target class is not significantly lower). Thus, relying on *MPC* to detect GTA tends to give missing or incorrect results.

We provide a possible explanation. Intuitively, NC relies on the assumption that a trojan model creates a "shortcut" (*i.e.*, the trigger perturbation) for all the trigger-embedded inputs to reach the target class. However, this premise does not necessarily hold for GTA: given its adaptive nature, each individual graph may have a specific shortcut to reach the target class, rendering the detection less effective. It thus seems crucial to carefully account for the trigger adaptiveness in designing countermeasures against GTA.

5.3 Input-space attacks

While GTA directly operates on graph-structured inputs, there are scenarios in which non-graph inputs are converted to graphs for GNNs to process. In this case, the adversary must ensure that any perturbation on the graph after applying the trigger can be realistically projected back to the input space, where the adversary performs the manipulation. Although input-space attacks are an ongoing area of research [44], here we discuss the challenges and potential solutions to the problem for graph-structure data.

Challenges and solutions – Let $\mathcal{X}$ and $\mathcal{G}$ be the input and graph spaces, π be the transformation mapping an input $X \in \mathcal{X}$ to its graph $G \in \mathcal{G}$, and r and δ be the corresponding perturbations in the input and graph spaces, respectively. To implement GTA in the input space, the adversary needs to (*i*) find r corresponding to given δ and (*ii*) ensure that r satisfies the semantic constraints ρ of the input space (*e.g.*, malware retains its malicious functionality). We temporarily assume it is feasible to find r for given δ and focus on enforcing r to satisfy the input-space constraint ρ.

Transferable constraint – In the case that ρ directly applies to the graph space, we may constrain r to be the transplantation of syntactically-equivalent benign ASTs. For example, we may craft malicious JavaScripts (input space) using ASTs (graph space) [15] taken from benign samples.

Specifically, we define a function $\rho(G)$ to measure G's compliance with ρ. We differentiate two cases. First, if ρ is differentiable (*e.g.*, modeled as GNN [68]), we define a regularizer in training g_t (*cf.* Eq (2)):

$$\ell_{\text{reg}}(g_t) = \mathbb{E}_{G \in \mathcal{D}_{[y_t]}} \Delta(\rho(G), \rho(m(G; g_t))) \tag{13}$$

where Δ measures the difference of the compliance of two graphs G and $m(G; g_t)$. Second, if ρ is non-differentiable, we restrict δ to perturbations guaranteed to satisfy ρ. For instance, to preserve the functionality of a malicious program, we may add edges corresponding to no-op calls in its CFG.

Non-transferable constraint – In the case that ρ is inapplicable to the graph space, it is infeasible to directly check δ's validity. For instance, it is difficult to check the tree structure of a PDF malware to determine whether it preserves the malicious network functionality [70].

We consider two cases. (*i*) If π's inversion π^{-1} and ρ are differentiable, we define a regularizer in training g_t (*cf.* Eq (2)):

$$\ell_{\text{reg}}(g_t) = \mathbb{E}_{G \in \mathcal{D}_{[y_t]}} \rho\left(\pi^{-1}(m(G; g_t))\right) \tag{14}$$

(*ii*) If π^{-1} or ρ is non-differentiable, one may use a problem-driven search strategy. Specifically, the search starts with a random mutation δ and learns from experience how to appropriately mutate it to satisfy ρ and the objectives in Eq (3). To implement this strategy, it requires to re-design GTA within a reinforcement learning framework.

Case study – Here, we conduct a case study of input-space GTA in the task of detecting malicious Android APKs [1].

Attack Setting	Effectiveness (ASR|AMC)		Evasiveness (CAD)	
	input-space	graph-space	input-space	graph-space
Topology Only	94.3% .952	97.2% .977	0.9%	0.0%
Topology + Feature	96.2% .971	100% .980	1.9%	0.9%

Table 9. Comparison of input-space and graph-space GTA. Topology-only – node features are defined as occurrences of API names; only topology perturbation is allowed. Topology & Feature – node features are defined as detailed call features; both topology and feature perturbations are allowed.

We conduct input-space GTA as a repackaging process [51], which converts a given APK X to its call-graph G (using Soot[4]), injects the trigger into G to form another graph $\tilde{G}$, and converts $\tilde{G}$ back to a perturbed APK $\tilde{X}$ (using Soot). Yet, the perturbation to G must ensure that it is feasible to find $\tilde{X}$ corresponding to $\tilde{G}$, while $\tilde{X}$ preserves X's functionality. We thus define the following perturbation:

The perturbation to G's topological structures is limited to adding no-op calls. Specifically, we define a binary mask matrix M_{msk} and set its ij-th entry as 1 if (*i*) A_{ij} is 1, which retains the original call, or (*ii*) node i is an external method, which is controlled by the adversary, and node j is either an external method or an internal read-only method, which does not influence the original functionality.

The perturbation to G's node features is limited to the modifiable features in Table 11 and constrained by their semantics. Specifically, we use the 23∼37-th features, which correspond to the call frequencies of 15 specific instructions and only increase their values, implementable by adding calls of such instructions during the repackaging process. Here, we focus on showing the feasible of input-space attacks, while admitting the possibility that such no-op calls could be potentially identified and removed via decompiling the APK file.

Results and analysis – We evaluate input-space GTA on the AndroZoo dataset (*cf.* Table 1), which is partitioned into 50%/50% for the pre-training and downstream tasks, respectively. We use a GCN as the feature extractor and a FCN as the classifier (*cf.* Table 10). We consider two settings. (*i*) Each node is associated with a one-hot vector, each dimension corresponding to one key API of fundamental importance to malware detection [19]. Under this setting, the attack is only allowed to perturb topological structures. (*ii*) Each node is associated with 40 call features (*cf.* Table 11). Under this setting, the attack is allowed to perturb both topological structures and node features. The system built upon benign GNNs achieves 95.3% and 98.1% *ACC* under the two settings, respectively.

We implement input-space GTA and compare it with graph-space GTA unbounded by input-space constraints. The results are summarized in Table 9. Under both settings, input-space and graph-space attacks attain high effectiveness (with *ASR* above 94% and *AMC* over 0.95), effectively retain the accuracy of benign GNNs (with *CAD* below 2%). As expected, due to its additional semantic constraints, input-space GTA performs worse than graph-space GTA in terms of both effectiveness and evasiveness; yet, because of the adaptive nature of GTA,

[4] Soot: https://github.com/soot-oss/soot

the constraints have a limited impact (e.g., less than 4% lower in *ASR* and less than 1% higher in *CAD*).

We further manually inspect the APKs repackaged by input-space GTA to verify the correctness of the perturbations: (*i*) we install the APK on an Android device and test its functionality; (*ii*) we apply Soot to convert the repackaged APK back to its call-graph and check whether all the injected calls are successfully retained; (*iii*) we trigger the methods where the injected calls originate to check whether the app crashes or whether there are warnings/errors in the system logs. With the manual inspection of the repackaged APKs, we find all the input-space perturbations satisfy (*i*), (*ii*), and (*iii*).

Limitations – Although the case study above demonstrates an example of input-space GTA, there are still limitations that may impact its feasibility in certain settings, which we believe offers several interesting avenues for future research. First, it may be inherently infeasible to modify the input to achieve the desirable perturbation in the graph space. For instance, it is often assumed difficult to directly modify biometric data (*e.g.*, fingerprints). Further, there are cases in which it is impractical to model the semantic constraints, not to mention using them to guide the attack. For instance, it is fundamentally difficult to verify the existence of chemical compounds corresponding to given molecular graphs [74]. Finally, the adversary may only have limited control over the input. For instance, the adversary may only control a small number of accounts in a social network such as Facebook, while the perturbation (*e.g.*, adding fake relationships) may be easily nullified by the network's dynamic evolution.

6 Related Work

With their wide use in security-critical domains, DNNs become the new targets of malicious manipulations [3]. Two primary types of attacks are considered in the literature.

Adversarial attacks – One line of work focuses on developing new attacks of crafting adversarial inputs to deceive target DNNs [6, 20, 43, 55]. Another line of work attempts to improve DNN resilience against existing attacks by devising new training strategies (*e.g.*, adversarial training) [22, 29, 42, 57] or detection methods [18, 36, 39, 69]. However, such defenses are often penetrated or circumvented by even stronger attacks [2, 31], resulting in a constant arms race.

Backdoor attacks – The existing backdoor attacks can be classified based on their targets. In class-level attacks, specific triggers (*e.g.*, watermarks) are often pre-defined, while the adversary aims to force all the trigger-embedded inputs to be misclassified by the trojan model [21, 34]. In instance-level attacks ("clean-label" backdoors), the targets are pre-defined, unmodified inputs, while the adversary attempts to force such inputs to be misclassified by the trojan model [26, 27, 50, 54]. The existing defenses against backdoor attacks mostly focus on class-level attacks, which, according to their strategies, include (*i*) cleansing potential contaminated data at training time [58], (*ii*) identifying suspicious models during model inspection [9, 33, 60], and (*iii*) detecting trigger-embedded inputs at inference [7, 11, 13, 17].

Attacks against GNNs – In contrast of the intensive research on general DNNs, the studies on the security properties of GNNs for graph-structured data are still sparse. One line of work attempts to deceive GNNs via perturbing the topological structures or descriptive features of graph data at inference time [12, 61, 77]. Another line of work aims to poison GNNs during training to degrade their overall performance [4, 32, 78]. The defenses [63, 67] against such attacks are mostly inspired by that for general DNNs (*e.g.*, adversarial training [37]).

Despite the plethora of prior work, the vulnerabilities of GNNs to backdoor attacks are largely unexplored. Concurrent to this work, Zhang *et al.* [75] propose a backdoor attack against GNNs via training trojan GNNs with respect to pre-defined triggers. This work differs in several major aspects: (*i*) considering both inductive and transductive tasks, (*ii*) optimizing both triggers and trojan models, and (*iii*) exploring the effectiveness of state-of-the-art backdoor defenses.

7 Conclusion

This work represents an in-depth study on the vulnerabilities of GNN models to backdoor attacks. We present GTA, the first attack that trojans GNNs and invokes malicious functions in downstream tasks via triggers tailored to individual graphs. We showcase the practicality of GTA in a range of security-critical applications, raising severe concerns about the current practice of re-using pre-trained GNNs. Moreover, we provide analytical justification for such vulnerabilities and discuss potential mitigation, which might shed light on pre-training and re-using GNNs in a more secure fashion.

Acknowledgments

We thank our shepherd Scott Coull and anonymous reviewers for their constructive feedback. This work is supported by the National Science Foundation under Grant No. 1951729, 1953813, and 1953893. Any opinions, findings, and conclusions or recommendations are those of the authors and do not necessarily reflect the views of the National Science Foundation. Shouling Ji was partly supported by the National Key Research and Development Program of China under No. 2018YFB0804102 and No. 2020YFB2103802, NSFC under No. 61772466, U1936215, and U1836202, the Zhejiang Provincial Natural Science Foundation for Distinguished Young Scholars under No. LR19F020003, and the Fundamental Research Funds for the Central Universities (Zhejiang University NGICS Platform).

References

[1] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. AndroZoo: Collecting Millions of Android Apps for the Research Community. In *Proceedings of Conference on Mining Software Repositories (MSR)*, 2016.

[2] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples. In *Proceedings of IEEE Conference on Machine Learning (ICML)*, 2018.

[3] Battista Biggio and Fabio Roli. Wild Patterns: Ten Years after The Rise of Adversarial Machine Learning. *Pattern Recognition*, 84:317–331, 2018.

[4] Aleksandar Bojchevski and Stephan Günnemann. Adversarial Attacks on Node Embeddings via Graph Poisoning. In *Proceedings of IEEE Conference on Machine Learning (ICML)*, 2019.

[5] BVLC. ModelZoo. https://github.com/BVLC/caffe/wiki/Model-Zoo, 2017.

[6] Nicholas Carlini and David A. Wagner. Towards Evaluating the Robustness of Neural Networks. In *Proceedings of IEEE Symposium on Security and Privacy (S&P)*, 2017.

[7] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting Backdoor Attacks on Deep Neural Networks by Activation Clustering. In *ArXiv e-prints*, 2018.

[8] 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.

[9] Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. DeepInspect: A Black-box Trojan Detection and Mitigation Framework for Deep Neural Networks. In *Proceedings of Joint Conference on Artificial Intelligence*, 2019.

[10] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning. *ArXiv e-prints*, 2017.

[11] Edward Chou, Florian Tramer, Giancarlo Pellegrino, and Dan Boneh. SentiNet: Detecting Physical Attacks Against Deep Learning Systems. In *ArXiv e-prints*, 2018.

[12] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial Attack on Graph Structured Data. In *Proceedings of IEEE Conference on Machine Learning (ICML)*, 2018.

[13] Bao Doan, Ehsan Abbasnejad, and Damith Ranasinghe. Februus: Input Purification Defense Against Trojan Attacks on Deep Neural Network Systems. In *ArXiv e-prints*, 2020.

[14] Elliptic. www.elliptic.co.

[15] Aurore Fass, Michael Backes, and Ben Stock. HideNoSeek: Camouflaging Malicious JavaScript in Benign ASTs. In *Proceedings of ACM Conference on Computer and Communications (CCS)*, 2019.

[16] Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimilano Pontil. Bilevel Programming for Hyperparameter Optimization and Meta-Learning. In *Proceedings of IEEE Conference on Machine Learning (ICML)*, 2018.

[17] Yansong Gao, Chang Xu, Derui Wang, Shiping Chen, Damith Ranasinghe, and Surya Nepal. STRIP: A Defence Against Trojan Attacks on Deep Neural Networks. In *ArXiv e-prints*, 2019.

[18] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *Proceedings of IEEE Symposium on Security and Privacy (S&P)*, 2018.

[19] Liangyi Gong, Zhenhua Li, Feng Qian, Zifan Zhang, Qi Alfred Chen, Zhiyun Qian, Hao Lin, and Yunhao Liu. Experiences of Landing Machine Learning onto Market-Scale Mobile Malware Detection. In *Proceedings of European Conference on Computer Systems (EuroSys)*, 2020.

[20] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. In *Proceedings of Conference on Learning Representations (ICLR)*, 2015.

[21] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain. *ArXiv e-prints*, 2017.

[22] Chuan Guo, Mayank Rana, Moustapha Cissé, and Laurens van der Maaten. Countering Adversarial Images Using Input Transformations. In *Proceedings of Conference on Learning Representations (ICLR)*, 2018.

[23] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[24] William L. Hamilton, Rex Ying, and Jure Leskovec. Representation Learning on Graphs: Methods and Applications. *IEEE Data Engineering Bulletin*, 3(40):52–74, 2017.

[25] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for Pre-training Graph Neural Networks. In *Proceedings of Conference on Learning Representations (ICLR)*, 2020.

[26] Yujie Ji, Xinyang Zhang, Shouling Ji, Xiapu Luo, and Ting Wang. Model-Reuse Attacks on Deep Learning Systems. In *Proceedings of ACM Conference on Computer and Communications (CCS)*, 2018.

[27] Yujie Ji, Xinyang Zhang, and Ting Wang. Backdoor Attacks against Learning Systems. In *Proceedings of IEEE Conference on Communications and Network Security (CNS)*, 2017.

[28] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of Conference on Learning Representations (ICLR)*, 2017.

[29] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial Machine Learning at Scale. In *Proceedings of Conference on Learning Representations (ICLR)*, 2017.

[30] Shaofeng Li, Benjamin Zi Hao Zhao, Jiahao Yu, Minhui Xue, Dali Kaafar, and Haojin Zhu. Invisible Backdoor Attacks Against Deep Neural Networks. *ArXiv e-prints*, 2019.

[31] X. Ling, S. Ji, J. Zou, J. Wang, C. Wu, B. Li, and T. Wang. DEEPSEC: A Uniform Platform for Security Analysis of Deep Learning Model. In *Proceedings of IEEE Symposium on Security and Privacy (S&P)*, 2019.

[32] Xuanqing Liu, Si Si, Xiaojin Zhu, Yang Li, and Cho-Jui Hsieh. A Unified Framework for Data Poisoning Attack to Graph-based Semi-supervised Learning. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[33] 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 ACM Conference on Computer and Communications (CCS)*, 2019.

[34] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. In *Proceedings of Network and Distributed System Security Symposium (NDSS)*, 2018.

[35] Yunfei Liu, Xingjun Ma, James Bailey, and Feng Lu. Reflection Backdoor: A Natural Backdoor Attack on Deep Neural Networks. In *Proceedings of European Conference on Computer Vision (ECCV)*, 2020.

[36] Shiqing Ma, Yingqi Liu, Guanhong Tao, Wen-Chuan Lee, and Xiangyu Zhang. NIC: Detecting Adversarial Samples with Neural Network Invariant Checking. In *Proceedings of Network and Distributed System Security Symposium (NDSS)*, 2019.

[37] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. In *Proceedings of Conference on Learning Representations (ICLR)*, 2018.

[38] Andreas Mayr, Günter Klambauer, Thomas Unterthiner, Marvin Steijaert, Jörg K. Wegner, Hugo Ceulemans, Djork-Arné Clevert, and Sepp Hochreiter. Large-Scale Comparison of Machine Learning Methods for Drug Target Prediction on ChEMBL. *Chem. Sci.*, 9:5441–5451, 2018.

[39] Dongyu Meng and Hao Chen. MagNet: A Two-Pronged Defense Against Adversarial Examples. In *Proceedings of ACM Conference on Computer and Communications (CCS)*, 2017.

[40] Michel Neuhaus and Horst Bunke. A Graph Matching Based Approach to Fingerprint Classification Using Directional Variance. *Lecture Notes in Computer Science*, 3546:455–501, 2005.

[41] Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(10):1367–1372, 2004.

[42] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks. In *Proceedings of IEEE Symposium on Security and Privacy (S&P)*, 2016.

[43] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The Limitations of Deep Learning in Adversarial Settings. In *Proceedings of IEEE European Symposium on Security and Privacy (Euro S&P)*, 2016.

[44] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. Intriguing Properties of Adversarial ML Attacks in the Problem Space. In *Proceedings of IEEE Symposium on Security and Privacy (S&P)*, 2019.

[45] Dimitris N. Politis, Joseph P. Romano, and Michael Wolf. *Subsampling*. Springer, 1999.

[46] Smita Ranveer and Swapnaja Hiray. Comparative Analysis of Feature Extraction Methods of Malware Detection. *Journal of Computer Applications*, 120(5), 2015.

[47] Ryan A. Rossi and Nesreen K. Ahmed. The Network Data Repository with Interactive Graph Analytics and Visualization. 2015.

[48] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale Attributed Node Embedding. In *ArXiv e-prints*, 2019.

[49] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. Hidden Technical Debt in Machine Learning Systems. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

[50] Ali Shafahi, W. Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison Frogs! Targeted Clean-Label Poisoning Attacks on Neural Networks. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[51] Yuru Shao, Xiapu Luo, Chenxiong Qian, Pengfei Zhu, and Lei Zhang. Towards a Scalable Resource-Driven Approach for Detecting Repackaged Android Applications. In *Proceedings of Annual Computer Security Applications Conference (ACSAC)*, 2014.

[52] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. TBT: Targeted Neural Network Attack with Bit Trojan. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[53] Wei Song, Heng Yin, Chang Liu, and Dawn Song. DeepMem: Learning Graph Neural Network Models for Fast and Robust Memory Forensic Analysis. In *Proceedings of ACM Conference on Computer and Communications (CCS)*, 2018.

[54] Octavian Suciu, Radu Mărginean, Yiğitcan Kaya, Hal Daumé, III, and Tudor Dumitraş. When Does Machine Learning FAIL? Generalized Transferability for Evasion and Poisoning Attacks. In *Proceedings of USENIX Security Symposium (SEC)*, 2018.

[55] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing Properties of Neural Networks. In *Proceedings of Conference on Learning Representations (ICLR)*, 2014.

[56] Tox21 Data Challenge. https://tripod.nih.gov/tox21/, 2014.

[57] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel. Ensemble Adversarial Training: Attacks and Defenses. In *Proceedings of Conference on Learning Representations (ICLR)*, 2018.

[58] Brandon Tran, Jerry Li, and Aleksander Madry. Spectral Signatures in Backdoor Attacks. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[59] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. In *Proceedings of Conference on Learning Representations (ICLR)*, 2018.

[60] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao. Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks. In *Proceedings of IEEE Symposium on Security and Privacy (S&P)*, 2019.

[61] Binghui Wang and Neil Zhenqiang Gong. Attacking Graph-based Classification via Manipulating the Graph Structure. In *Proceedings of ACM Conference on Computer and Communications (CCS)*, 2019.

[62] Binghui Wang, Jinyuan Jia, and Neil Zhenqiang Gong. Graph-based Security and Privacy Analytics via Collective Classification with Joint Weight Learning and Propagation. In *Proceedings of Network and Distributed System Security Symposium (NDSS)*, 2019.

[63] Shen Wang, Zhengzhang Chen, Jingchao Ni, Xiao Yu, Zhichun Li, Haifeng Chen, and Philip S. Yu. Adversarial Defense Framework for Graph Neural Network. In *ArXiv e-prints*, 2019.

[64] Shen Wang, Zhengzhang Chen, Xiao Yu, Ding Li, Jingchao Ni, Lu-An Tang, Jiaping Gui, Zhichun Li, Haifeng Chen, and Philip S. Yu. Heterogeneous Graph Matching Networks for Unknown Malware Detection. 2019.

[65] C.I. Watson and C.L. Wilson. *NIST Special Database 4, Fingerprint Database*. National Institute of Standards and Technology, 1992.

[66] B. Yu. Weisfeiler and A. A. Leman. Reduction of A Graph to A Canonical Form and An Algebra Arising during This Reduction. *Nauchno-Technicheskaya Informatsia*, 2:12–16, 1968.

[67] 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. 2019.

[68] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks? In *Proceedings of Conference on Learning Representations (ICLR)*, 2019.

[69] W. Xu, D. Evans, and Y. Qi. Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks. In *Proceedings of Network and Distributed System Security Symposium (NDSS)*, 2018.

[70] Weilin Xu, Yanjun Qi, and David Evans. Automatically Evading Classifiers: A Case Study on PDF Malware Classifiers. In *Proceedings of Network and Distributed System Security Symposium (NDSS)*, 2016.

[71] Yuanshun Yao, Huiying Li, Haitao Zheng, and Ben Y. Zhao. Latent Backdoor Attacks on Deep Neural Networks. In *Proceedings of ACM Conference on Computer and Communications (CCS)*, 2019.

[72] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical Graph Representation Learning with Differentiable Pooling. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[73] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How Transferable Are Features in Deep Neural Networks? In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

[74] Chengxi Zang and Fei Wang. MoFlow: An Invertible Flow Model for Generating Molecular Graphs. In *Proceedings of ACM Conference on Knowledge Discovery and Data Mining (KDD)*, 2020.

[75] Zaixi Zhang, Jinyuan Jia, Binghui Wang, and Neil Zhenqiang Gong. Backdoor Attacks to Graph Neural Networks. In *ArXiv e-prints*, 2020.

[76] Marinka Zitnik, Rok Sosič, and Jure Leskovec. Prioritizing Network Communities. *Nature Communications*, 9(1):2544, 2018.

[77] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial Attacks on Neural Networks for Graph Data. In *Proceedings of ACM Conference on Knowledge Discovery and Data Mining (KDD)*, 2018.

[78] Daniel Zügner and Stephan Günnemann. Adversarial Attacks on Graph Neural Networks via Meta Learning. In *Proceedings of Conference on Learning Representations (ICLR)*, 2019.

A Implementation details

A.1 Look-ahead step

To evaluate Eq (5), we apply the chain rule:

$$\nabla_{g_t} \ell_{\text{atk}}\left(\theta', g_t\right) - \xi \nabla^2_{g_t,\theta} \ell_{\text{ret}}\left(\theta, g_t\right) \nabla_{\theta'} \ell_{\text{atk}}\left(\theta', g_t\right) \tag{15}$$

where $\theta' = \theta - \xi \nabla_\theta \ell_{\text{ret}}\left(\theta, g_t\right)$ is the updated parameter after the one-step look-ahead. This formulation involves matrix-vector multiplication, which can be approximated with the finite difference approximation. Let $\theta^{\pm} = \theta \pm \varepsilon \nabla_{\theta'} \ell_{\text{atk}}\left(\theta', g_t\right)$ where ε is a small constant (*e.g.*, $\varepsilon = 10^{-5}$). We can approximate the second term of Eq (15) as:

$$\frac{\nabla_{g_t} \ell_{\text{ret}}\left(\theta^+, g_t\right) - \nabla_{g_t} \ell_{\text{ret}}\left(\theta^-, g_t\right)}{2\varepsilon} \tag{16}$$

A.2 Mixing function

The mixing function $m(G; g_t)$ specifies how trigger g_t is embedded into graph G by replacing subgraph g in G with g_t. We extend a backtracking-based algorithm VF2 [41] to search for g most similar to g_t. Intuitively, VF2 recursively extends a partial match by mapping the next node in g_t to a node in G; if it is feasible, it extends the partial match and recurses, and backtracks otherwise. As we search for the most similar subgraph, we maintain the current highest similarity and terminate a partial match early if it exceeds this threshold. Algorithm 2 sketches the implementation of the mixing function.

A.3 Transductive attack

Algorithm 3 sketches the implementation of GTA in transductive tasks (*e.g.*, node classification).

Algorithm 2: Mixing function $m(G; g_t)$

Input: g_t - trigger subgraph; G - target graph;
Output: g - subgraph in G to be replaced

```
// initialization
c_best ← ∞, M ← ∅, g_best ← ∅;
specify a topological order v_0, v_1, ..., v_{n-1} over g_t;
foreach node u in G do
    add (u, v_0) to M;
    while M ≠ ∅ do
        (u_j, v_i) ← top pair of M;
        if i = n − 1 then
            // all nodes in g_t covered by M
            compute M's distance as c_cur;
            if c_cur < c_best then c_best ← c_cur, g_best ← G's part in M ;
            pop top pair off M;
        else
            if there exists extensible pair (u_k, v_{i+1}) then
                if M's distance < c_best then add (u_k, v_{i+1}) to M;
            else pop top pair off M;
return g_best;
```

Algorithm 3: GTA (transductive) attack

Input: $\theta_\circ$ - pre-trained GNN; G - target graph; y_t - target class;
Output: θ - trojan GNN; ω - parameters of trigger generation function

```
// initialization
randomly initialize ω;
randomly sample subgraphs {g} ∼ G;
// bi-level optimization
while not converged yet do
    // updating trojan GNN
    update θ by descent on ∇_θ ℓ_ret(θ, g_t) (cf. Eq (9));
    // updating trigger generation function
    update ω by descent on ∇_ω ℓ_atk(θ − ξ∇_θ ℓ_ret(θ, g_t), g_t) (cf. Eq (8));
return (θ, ω);
```

A.4 Parameter setting

Table 10 summarizes the default parameter setting .

Type	Parameter	Setting
GCN	Architecture	2AL
GRAPHSAGE	Architecture	2AL
	Aggregator	Mean [23]
GCN (off-the-shelf)	Architecture	5AL
GAT	# Heads	3
Classifier	Architecture	FCN (1FC+1SM)
Training	Optimizer	Adam
	Learning rate	0.01
	Weight decay	5e-4
	Dropout	0.5
	Epochs	50 (*I*), 100 (*T*)
	Batch size	32 (*I*)
Attack	n_{trigger}	3 (*I*), 6 (*T*)
	n_{io}	1
	n_{mask}	100% (*I*), 10% (*T*)
	n_{iter}	3
Trigger Generator	Optimizer	Adam
	Learning rate	0.01
	Epochs	20
Detection	# Samples	100 per class
	Significance level α	0.05
	λ_{ASR}	80%

Table 10. Default parameter setting. I: inductive, T: transductive, AL: aggregation layer, FC: fully-connected layer, SM: softmax layer.

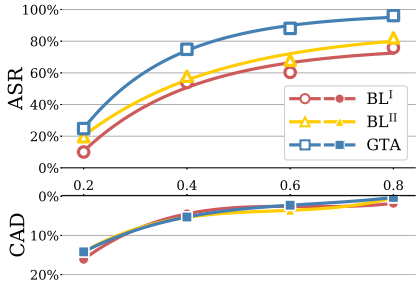


Figure 13: Attack effectiveness and evasiveness of GTA with respect to varying subsampling ratio β.

Results and analysis – We evaluate RS in the transfer case of ChEMBL→Toxicant (*cf.* Table 4). Figure 13 illustrates the effectiveness and evasiveness of GTA as a function of the subsampling ratio β. Observe that there exists an intricate trade-off between attack robustness and clean accuracy. A smaller β leads to lower *ASR* but also results in larger *CAD*. Therefore, RS may not be a viable option for defending against GTA, as it may negatively impact system performance.

Other input-inspection defenses – One may suggest using other input inspection methods. Yet, it is often challenging to extend such defenses from continuous domains (*e.g.*, images) to discrete domains (*e.g.*, graphs). For instance, STRIP [13] is a representative input inspection defense. Intuitively, if an input is embedded with a trigger, its mixture with a benign input is still dominated by the trigger and tends to be misclassified to the target class, resulting in relatively low entropy of the prediction. Unfortunately, it is intrinsically difficult to apply STRIP to graph-structured data. For instance, it is challenging to meaningfully "mix" two graphs.

B Additional experiments

B.1 Input inspection as a defense

We build our defense upon Randomized-Smoothing (RS) [75].

Randomized smoothing – RS applies a subsampling function $\mathcal{S}$ over a given graph G (including both its structural connectivity and node features), generates a set of subsampled graphs $G_1, G_2, \ldots, G_n$, and takes a majority voting of the predictions over such samples as G's final prediction. Intuitively, if G is trigger-embedded, the trigger is less likely to be effective on the subsampled graphs, due to their inherent randomness. In particular, $\mathcal{S}$ is controlled by a parameter β (subsampling ratio), which specifies the randomization magnitude. For instance, if $\beta = 0.8$, $\mathcal{S}$ randomly removes 20% of G's nodes, and for the rest nodes, randomly sets 20% of their features to be 0. Note that while in [75], RS is further extended to mitigate trojan GNNs, here we focus on its use as a defense against trigger-embedded graphs.

B.2 Input-space attacks

Table 11 summarizes 40 features associated with each node in the Android call graphs.

Feature #	Definition	Value	Constraint
0	number of parameters	$[0,20]$	increasing only, requiring to modify 1∼20 accordingly
1∼20	parameter type (*e.g.*, 'int')	$[0,23]$	subtype to supertype only
21	return type	$[0,21]$	same as above
22	modifier (*e.g.*, 'private')	$[0,4]$	increasing only
23∼37	instruction call frequency	$[0,\infty]$	increasing only
38	affiliation type	$[0,\infty]$	non-modifiable
39	package name	$[0,\infty]$	non-modifiable

Table 11. Descriptive features of Android call graphs and corresponding perturbation constraints.

Figure 14 visualizes sample call graphs generated by input-space GTA, where only external methods are perturbed.

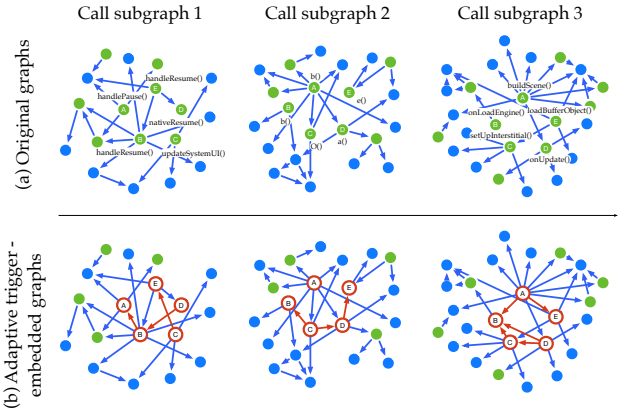


Figure 14: Illustration of input-space GTA on Android call graphs: green node – external method; blue node – Android internal method; red node – method perturbed by GTA; blue edge – original call; red edge – no-op call added by GTA. Only the vicinity of the perturbed subgraph is shown.

C Graph-space constraints

We consider two types of constraints specified respectively on G's topological connectivity and node features respectively.

Topological structures – Let g be the subgraph in G to be replaced by the trigger g_t. Let A denote g's adjacency matrix with A_{ij} indicating whether nodes i, j are connected. Recall that to generate g_t, GTA computes another adjacency matrix $\tilde{A}$ as in Eq (6) and replaces A with $\tilde{A}$. We consider the following constraints over this operation.

- Presence/absence of specific edges, which excludes certain pairs of nodes from perturbation. We define the perturbation as: $M_{msk} \odot A + (1 - M_{msk}) \odot \tilde{A}$, where M_{msk} is a binary mask matrix and $\odot$ denotes element-wise multiplication. Intuitively, A_{ij} is retained if the ij-th entry of M_{msk} is on and replaced by $\tilde{A}_{ij}$ otherwise.
- Addition/deletion only which specifies whether only adding/removing edges is allowed. To enforce the addition-only constraint (similar in the case of deletion only), we set the ij-th entry of M_{msk} to be 1 if A_{ij} is 1 and 0 otherwise, which retains all the edges in g.
- Perturbation magnitude, which limits the number of perturbed edges. To enforce the constraint, we add a regularizer $\|A - \tilde{A}\|_F$ to the objective function in Eq (3), where $\|\cdot\|_F$ denotes the Frobenius norm. Intuitively, the regularizer penalizes a large perturbation from A to $\tilde{A}$.
- (Sub)graph isomorphism, which dictates the isomorphism of g and g_t (extensible to other (sub)graphs of G and $m(G; g_t)$). To enforce this constraint, we add a regularizer $\Delta(\rho(A), \rho(\tilde{A}))$ to the objective function in Eq (3), where ρ maps a graph to its encoding for isomorphism testing and Δ measures the difference between two encodings. In particular, ρ can be modeled (approximately) as a GNN [68].

Node features – Recall that GTA replaces the feature vector X_i of each node $i \in g$ with its corresponding feature $\tilde{X}_i$ in g_t. We consider two types of constraints on this operation.

- Exclusion of specific features, which excludes certain features from perturbation. To improve the trigger evasiveness, we may restrict the replacement to certain features: $v_{\text{msk}} \odot X_i + (1 - v_{\text{msk}}) \odot \tilde{X}_i$, where the mask v_{msk} is a binary vector and $\odot$ denotes element-wise multiplication. Intuitively, the j-th feature of $\tilde{X}_i$ is retained if the j-th bit of v_{msk} is on and replaced by the j-th feature of $\tilde{X}_i$ otherwise.
- Perturbation magnitude, which limits the number of perturbed features. To enforce this constraint, we consider the binary mask v_{msk} as a variable and limit the cardinality of v_{msk} by adding a regularizer $\|v_{\text{msk}}\|_1$ to the objective function in Eq (3), where $\|\cdot\|_1$ denotes the ℓ_1 norm.