Unnoticeable Backdoor Attacks on Graph Neural Networks

Enyan Dai*
emd5759@psu.edu
The Pennsylvania State University
State College, USA

Minhua Lin*
mfl5681@psu.edu
The Pennsylvania State University
State College, USA

Xiang Zhang
xzz89@psu.edu
The Pennsylvania State University
State College, USA

Suhang Wang
szw494@psu.edu
The Pennsylvania State University
State College, USA

ABSTRACT

Graph Neural Networks (GNNs) have achieved promising results in various tasks such as node classification and graph classification. Recent studies find that GNNs are vulnerable to adversarial attacks. However, effective backdoor attacks on graphs are still an open problem. In particular, backdoor attack poisons the graph by attaching triggers and the target class label to a set of nodes in the training graph. The backdoored GNNs trained on the poisoned graph will then be misled to predict test nodes to target class once attached with triggers. Though there are some initial efforts in graph backdoor attacks, our empirical analysis shows that they may require a large attack budget for effective backdoor attacks and the injected triggers can be easily detected and pruned. Therefore, in this paper, we study a novel problem of unnoticeable graph backdoor attacks with limited attack budget. To fully utilize the attack budget, we propose to deliberately select the nodes to inject triggers and target class labels in the poisoning phase. An adaptive trigger generator is deployed to obtain effective triggers that are difficult to be noticed. Extensive experiments on real-world datasets against various defense strategies demonstrate the effectiveness of our proposed method in conducting effective unnoticeable backdoor attacks.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**.

KEYWORDS

Backdoor Attack, Graph Neural Networks

ACM Reference Format:
Enyan Dai, Minhua Lin, Xiang Zhang, and Suhang Wang. 2023. Unnoticeable Backdoor Attacks on Graph Neural Networks. In *Proceedings of the ACM Web Conference 2023 (WWW '23), April 30–May 04, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3543507.3583392

*Both authors contribute equally to this paper.

WWW '23, April 30–May 04, 2023, Austin, TX, USA

ACM ISBN 978-1-4503-9416-1/23/04…$15.00
https://doi.org/10.1145/3543507.3583392

1 INTRODUCTION

Graph-structured data are very pervasive in the real-world such as social networks [14], finance system [29], and molecular graphs [17]. Recently, graph neural networks (GNNs) [19, 27] have shown promising results in modeling graphs. Generally, GNNs adopt the message-passing mechanism [19, 34], which updates a node's representation by aggregating information from its neighbors. As a result, the node representations learned by GNNs can preserve node features, neighbors and local graph topology, which facilitate various tasks such as semi-supervised node classification and graph classification.

Despite their great success of GNNs, GNNs are vulnerable to adversarial attacks and many graph attack methods have been investigated to fool the target GNN models to achieve adversarial goals [12, 33, 45]. Specifically, due to the utilization of graph structure with the message-passing mechanism, attackers can deliberately manipulate the graph structures and/or node features to mislead the GNNs for adversarial attacks. For instance, Nettack [45] iteratively modifies the connectivity of node pairs within attack budget to reduce the classification accuracy of GNNs on target nodes. However, the majority of existing attacks focus on graph manipulation attacks that require to calculate the optimal edges to be added/deleted for each target node. This will result in unaffordable time and space complexity on large-scale datasets, i.e, $O(N^2)$ where N is the number of nodes [10, 12, 45]. In addition, manipulating links and attributes of existing nodes is impractical as these nodes/individuals are not controlled by the attacker [25].

To address the aforementioned issues, one promising direction is to develop backdoor attacks on graphs. Fig. 1 gives an illustration of backdoor attack on graphs, where a small set of nodes denoted as poisoned samples will be attached with triggers and assigned the label of target class. The model trained on the poisoned graph will link the trigger with the target class. As a result, the target nodes will be predicted as the target class once they are attached with the triggers during the inference phase. The trigger can be either predefined or obtained from trigger generator. *Firstly*, the computation cost of backdoor attacks is limited compared with graph manipulation attacks, which paves us a way for an efficient target attack on large-scale graphs. For predefined triggers, nearly no computation cost is required. When a trigger generator is adopted, optimizing the trigger generator only needs the gradients from the poisoned samples. *Secondly*, once the backdoor is injected to the target GNNs, the predictions on new target nodes can be easily controlled by attaching generated triggers instead of an additional optimization

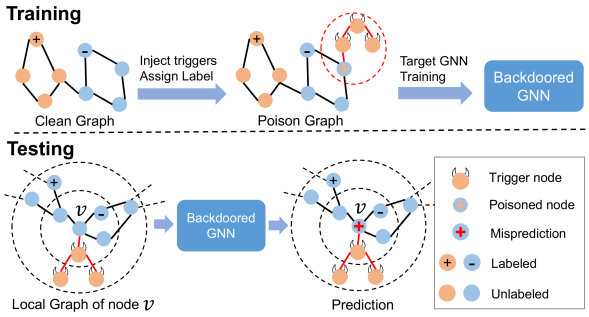


Figure 1: General framework of graph backdoor attack.

process as graph manipulation attacks. This will especially benefit the targeted attack on inductive node classification, which widely exists in real-world scenarios. For example, TikTok graph will often incorporate new users and predict labels of them with a trained model. *Thirdly*, compared with revising the links between existing users, it is relatively easy to inject triggers and malicious labels in backdoor attacks. Take malicious user detection on social networks as an example, many labels are collected from reports of users. In this case, malicious labels could be easily assigned by attackers. As for the trigger attachment, it can be achieved by linking a set of fake accounts to the users.

Recently, Zhang et al. [39] firstly investigate a graph backdoor attack that uses randomly generated graphs as triggers. A trigger generator is adopted in [30] to get more powerful sample-specific triggers. However, these methods have unnoticeablity issues in the following two aspects. *Firstly*, our empirical analysis in Sec. 3.3.1 shows that existing methods need a large budget to conduct effective backdoor attacks on large-scale graphs, i.e., they need to attach the backdoor triggers to a large number of nodes in the training graph so that a model trained on the graph will be fooled to assign target label to nodes attached with the backdoor trigger. This largely increases the risk of being detected. *Secondly*, the generated triggers of these methods can be easily identified and destroyed. Specifically, real-world graphs such as social networks generally follow homophily assumption, i.e., similar nodes are more likely to be connected; while in existing graph backdoor attacks, the edges linking triggers and poisoned nodes and edges inside the triggers are not guaranteed with the property of connecting nodes with high similarity scores. Thus, the triggers and assigned malicious labels can be eliminated by pruning edges linking dissimilar nodes and discarding labels of involved nodes, which is verified in Sec 3.3.2. Thus, developing an effective unnoticeable graph backdoor attack with limited attack budget is important. However, graph backdoor attack is still in its early stage and there is no existing work on unnoticeable graph backdoor attack with limited attack budget.

Therefore, in this paper, we study a novel and important problem of developing an effective unnoticeable graph backdoor attack with limited attack budget in terms of the number of poisoned nodes. In essence, we are faced with two challenges: (i) how to fully utilize the limited budget in poisoned samples for graph backdoor attacks; (ii) how to obtain triggers that are powerful and difficult to be detected. In an attempt to address these challenges, we proposed a novel framework Unnoticeable Graph Backdoor Attack (UGBA)[1].

[1]https://github.com/ventr1c/UGBA

To better utilize the attack budget, UGBA proposes to attach triggers with crucial representative nodes with a novel poisoned node selection algorithm. And an adaptive trigger generator is deployed in UGBA to obtain powerful unnoticeable trigger that exhibits high similarity with each target node and maintains high attack success rate. In summary, our main contributions are:

- We study a novel problem of promoting unnoticeablity of graph backdoor attacks in generated triggers and attack budget;
- We empirically verify that a simple strategy of edge pruning and label discarding can largely degrade existing backdoor attacks;
- We design a framework UGBA that deliberately selects poisoned samples and learn effective unnoticeable triggers to achieve unnoticeable graph backdoor attack under limited budget; and
- Extensive experiments on large-scale graph datasets demonstrate the effectiveness of our proposed method in unnoticeably backdooring different GNN models with limited attack budget.

2 RELATED WORKS

2.1 Graph Neural Networks

Graph Neural Networks (GNNs) [2, 19, 27, 35] have shown remarkable ability in modeling graph-structured data, which benefits various applications such as recommendation system [35], drug discovery [2] and traffic analysis [41]. Generally, the success of GNNs relies on the message-passing strategy, which updates a node's representation by recursively aggregating and combining features from neighboring nodes. For instance, in each layer of GCN [19] the representations of neighbors and the center node will be averaged followed by a non-linear transformation such as ReLU. Recently, many GNN models are proposed to further improve the performance of GNNs [4, 6, 7, 10, 18, 37, 43]. For example, self-supervised GNNs [18, 22, 43] are investigated to reduce the need of labeled nodes. Works that improve fairness [8], robustness [6, 7] and explainability [9, 36, 40] of GNNs are explored. And GNN models for heterophilic graphs are also desgined [11, 31].

2.2 Attacks on Graph Neural Networks

According to the stages the attack occurs, adversarial attacks on GNNs can be divided into poisoning attack [25, 45, 46] and evasion attack [1, 3, 12, 26, 28, 33]. In poisoning attacks, the attackers aim to perturb the training graph before GNNs are trained such that a GNN model trained on the poisoned dataset will have a low prediction accuracy on test samples. For example, Nettack [45] employs a tractable surrogate model to conduct a targeted poisoning attack by learning perturbation against the surrogate model. Evasion attacks add perturbation in the test stage, where the GNN model has been well trained and cannot be modified by attackers. Optimizing the perturbation of graph structures by gradient descent [33] and reinforcement learnings [12, 21] have been explored. Evasion attacks through graph injection [20, 44] are also investigated.

Backdoor attacks are still rarely explored on GNNs [30, 39]. Backdoor attacks generally attach backdoor triggers to the training data and assign the target label to samples with trigger. Then a model trained on the poisoned data will be misled if backdoors are activated by the trigger-embedded test samples. Zhang et al. [39] propose a subgraph-based backdoor attack on GNNs by injecting randomly generated universal triggers to some training samples.

Xi et al. [30] adopt a trigger generator to learn to generate adaptive trigger for different samples. Sheng et al. [24] propose to select the nodes with high degree and closeness centrality. Xu and Picek [32] improve the unnoticeability by assigning triggers without change labels of poisoned samples. Our proposed method is inherently different from these methods as (i) we can generate unnoticeable adaptive triggers to simultaneously maintain the effectiveness and bypass the potential trigger detection defense based on feature similarity of linked nodes; (ii) we design a novel clustering-based node selection algorithm to further reduce the required attack budget.

3 PRELIMINARY ANALYSIS

In this section, we present preliminaries of backdoor attacks on graphs and show unnoticeablity issues of existing backdoor attacks.

3.1 Notations

We use $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ to denote an attributed graph, where $\mathcal{V} = \{v_1, \ldots, v_N\}$ is the set of N nodes, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges, and $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$ is the set of node attributes with $\mathbf{x}_i$ being the node attribute of v_i. $\mathbf{A} \in \mathbb{R}^{N \times N}$ is the adjacency matrix of the graph $\mathcal{G}$, where $\mathbf{A}_{ij} = 1$ if nodes v_i and v_j are connected; otherwise $\mathbf{A}_{ij} = 0$. In this paper, we focus on a semi-supervised node classification task in the inductive setting, which widely exists in real-world applications. For instance, GNNs trained on social networks often need to conduct predictions on newly enrolled users to provide service. Specifically, in inductive node classification, a small set of nodes $\mathcal{V}_L \subseteq \mathcal{V}$ in the training graph are provided with labels $\mathcal{Y}_L = \{y_1, \ldots, y_{N_L}\}$. The test nodes $\mathcal{V}_T$ are not covered in the training graph $\mathcal{G}$, i.e., $\mathcal{V}_T \cap \mathcal{V} = \emptyset$.

3.2 Preliminaries of Graph Backdoor Attacks

3.2.1 Threat Model. In this section, we introduce the threat model.
Attacker's Goal: The goal of the adversary is to mislead the GNN model to classify target nodes attached with the triggers as target class. Simultaneously, the attacked GNN model should behave normally for clean nodes without triggers attached.
Attacker's Knowledge and Capability: As the setting of most poisoned attacks, the training data of the target model is available for attackers. The information of the target GNN models including model architecture is unknown to the attacker. Attackers are capable of attaching triggers and labels to nodes within a budget before the training of target models to poison graphs. During the inference phase, attackers can attach triggers to the target test node.

3.2.2 General Framework of Graph Backdoor Attacks. The key idea of the backdoor attacks is to associate the trigger with the target class in the training data to mislead target models. As Fig. 1 shows, during the poisoning phase, the attacker will attach a trigger g to a set of poisoned nodes $\mathcal{V}_P \subseteq \mathcal{V}$ and assign $\mathcal{V}_P$ with target class label y_t, resulting a backdoored dataset. Generally, the poisoned node set $\mathcal{V}_P$ is randomly selected. The GNNs trained on the backdoored dataset will be optimized to predict the poisoned nodes $\mathcal{V}_P$ attached with the trigger g as target class y_t, which will force the target GNN to correlate the existence of the trigger g in neighbors with the target class. In the test phase, the attacker can attach the trigger g to a test node v to make v classified as the target class by backdoored GNN. Some initial efforts [30, 39] have been made for graph backdoor attacks. Specifically, SBA [39] directly injects designed sub-graphs as triggers. And GTA [30] adopts a trigger generator to learn optimal sample-specific triggers.

Table 1: Impacts of $|\mathcal{V}_P|$ to ASR (%) of backdoor attacks.

$\lvert\mathcal{V}_P\rvert$	80	240	400	800	2400
SBA-Samp	0.06	1.7	10.8	34.5	75.5
SBA-Gen	0.08	18.1	32.1	54.3	85.9
GTA	37.4	62.4	72.4	82.7	94.8

Table 2: Results of backdoor defense (Attack Success Rate (%) | Clean Accuracy (%)) on Ogb-arxiv dataset.

Defense	Clean	SBA-Samp	SBA-Gen	GTA
None	65.5	61.0 | 65.1	70.8 | 65.2	94.8 | 65.6
Prune	62.2	8.9 | 64.0	31.2 | 64.0	1.4 | 64.5
Prune+LD	62.6	3.2 | 64.0	15.3 | 63.8	0.04 | 64.1

3.3 Unnoticeability of Graph Backdoor Attacks

In this subsection, we analyze the unnoticeability of existing graph backdoor attacks in terms of the required number of poisoned samples and the difficulty of trigger detection.

3.3.1 Size of Poisoned Nodes. In backdoor attacks, a set of poisoned nodes $\mathcal{V}_P$ will be attached triggers and target class labels to conduct attacks. However, as large-scale graphs can provide abundant information for training GNNs, the attacker may need to inject a large number of triggers and malicious labels to mislead the target GNN to correlate the trigger with target class, which puts backdoor attack at the risk of being noticed. To verify this, we analyze how the size of poisoned nodes affects the attack success rate of the state-of-the-art graph backdoor attacks, i.e., SBA-Gene, SBA-Samp [39], and GTA [30] on a large node classification dataset, i.e., OGB-arxiv [15]. Detailed descriptions of these methods can be found in Sec. 6.1.2. We vary $|\mathcal{V}_P|$ as $\{80, 240, 800, 2400\}$. The size of trigger is limited to contain three nodes. The architecture of target model is GraphSage [14]. The attack success rate (ASR) results are presented in Tabble 1. From the table, we can observe that all methods especially SBA-Gen and SBA-Samp achieve poor attack results with limited budget such as 80 and 240 in $\mathcal{V}_P$. This is because (i) SBA-Gen and SBA-Samp utilize handcrafted triggers which is not effective; (ii) Though GTA uses learned sample-specific trigger, similar to SBA-Gen and SBA-Samp, the selection of poisoned nodes is random and the budget is not well utilized. Thus, it is necessary to develop graph backdoor attack methods that can generate effective triggers and fully exploit the attack budget.

3.3.2 Detection of Triggers. Real-world graphs such as social networks generally show homophily property, i.e, nodes with similar attributes are connected by edges. For existing backdoor attacks, the attributes of triggers may differ a lot from the attached poisoned nodes. The connections within trigger may also violate homophily property. Therefore, the negative effects of injected triggers and target labels might be reduced by eliminating edges linking dissimilar nodes and labels of involved nodes. To verify this, we evaluate two strategies to defend against backdoor attacks:

- **Prune**: We prune edges linking nodes with low cosine similarity. As edges created by the backdoor attacker may link dissimilar nodes, the trigger structure and attachment edge can be destroyed.
- **Prune+LD**: To reduce the influence of dirty labels of poisoned nodes, besides pruning, we also discard the labels of the nodes linked by dissimilar edges.

Experimental results on Ogb-arxiv with $|\mathcal{V}_P|$ set as 2400 are presented in Table 2. Other settings are the same as Sec. 3.3.1. For Prune and Prune+LD, the threshold is to filter out edges with lowest 10% cosine similarity scores. More results on other datasets can be found in Table 4. The accuracy of the backdoored GNN on clean test set is also reported in Table 2 to show how the defense strategies affect the prediction performance. Accuracy on a clean graph without any attacks is reported as reference. All the results are average scores of 5 runs. We can observe from Tab. 2 that (i) ASR drops dramatically with the proposed two strategies of prune and prune+LD; (ii) the impact of the proposed strategies on prediction accuracy is negligible. This demonstrates that the used triggers by existing backdoor attacks can be easily mitigated.

4 PROBLEM FORMULATION

Our preliminary analysis verifies that existing backdoor attacks (i) require a large attack budget on large datasets; and (ii) the injected triggers can be easily detected. To alleviate these two issues, we propose to investigate a novel unnoticeable graph backdoor attack problem that can unnoticeablely backdoor various target GNNs with limited attack budget. Specifically, we enhance the general graph backdoor attack model from the following two aspects.

Selection of Poisoned Nodes $\mathcal{V}_P$: In the attack model of current graph backdoor attacks, the poisoned node set $\mathcal{V}_P$ is randomly selected. However, in this way, it is likely the budget is wasted in some useless poisoned nodes. For example, the attacker may repeatedly poison nodes from the same cluster that have very similar pattern, which is unnecessary. Alternatively, to fully utilize the attack budget, we will deliberately select the most useful poisoned nodes $\mathcal{V}_P \subseteq \mathcal{V}$ in unnoticeable backdoor attack.

Unnoticeable Constraint on Triggers: As the preliminary analysis shows, dissimilarity among trigger nodes and poisoned nodes makes the attack easy to be detected. Hence, it is necessary to obtain adaptive triggers that are similar to the poisoned nodes or target nodes. In addition, edges within triggers should also be enforced to link similar nodes to avoid being damaged by pruning strategy. Such adaptive trigger can be given by an adaptive generator. Let $\mathcal{E}_B^i$ denote the edge set that contain edges inside trigger g_i and edge attaching trigger g_i and node v_i. The unnoticeable constraint on the generated adaptive triggers can be formally written as:

$$\min_{(u,v)\in\mathcal{E}_B^i} sim(u,v) \geq T, \tag{1}$$

where sim denotes the cosine similarity between node features and T is a relatively high threshold of the cosine similarity which can be tuned based on datasets.

In node classification with GNNs, the prediction is given based on the computation graph of the node. Thus, the clean prediction on node v_i can be written as $f_\theta(\mathcal{G}_C^i)$, where $\mathcal{G}_C^i$ denotes the clean computation graph of node v_i. For a node v_i attached with the adaptive trigger g_i, the predictive label will be given by $f_\theta(a(\mathcal{G}_C^i, g_i))$, where $a(\cdot)$ denotes the operation of trigger attachment. Then, with the above descriptions and notations in Sec 3.1. we can formulate the unnoticeable graph backdoor attack by:

Problem 1. *Given a clean attributed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ with a set of nodes $\mathcal{V}_L$ provided with labels $\mathcal{Y}_L$, we aim to learn an adaptive trigger generator $f_g : v_i \rightarrow g_i$ and effectively select a set of nodes $\mathcal{V}_P$ within budget to attach triggers and labels so that a GNN f trained on the poisoned graph will classify the test node attached with the trigger to the target class y_t by solving:*

$$\begin{aligned}
&\min_{\mathcal{V}_P,\theta_g} \sum_{v_i\in\mathcal{V}_U} l(f_{\theta^*}(a(\mathcal{G}_C^i, g_i)), y_t) \\
&s.t.\ \theta^* = \arg\min_\theta \sum_{v_i\in\mathcal{V}_L} l(f_\theta(\mathcal{G}_C^i), y_i) + \sum_{v_i\in\mathcal{V}_P} l(f_\theta(a(\mathcal{G}_C^i, g_i)), y_t), \\
&\quad \forall v_i \in \mathcal{V}_P \cup \mathcal{V}_T,\ g_i \text{ meets Eq.(1) and } |g_i| < \Delta_g \\
&\quad |\mathcal{V}_P| \leq \Delta_P
\end{aligned} \tag{2}$$

where $l(\cdot)$ represents the cross entropy loss and θ_g denotes the parameters of the adaptive trigger generator f_g. In the constraints, the node size of trigger $|g_i|$ is limited by Δ_g, and the size of poisoned nodes is limited by Δ_P. The architecture of the target GNN f is unavailable and may adapt various defense methods.

In transductive setting, $\mathcal{V}_U$ would be the target nodes. However, we focus on inductive setting where $\mathcal{V}_T$ is not available for the optimization. Hence, $\mathcal{V}_U$ would be $\mathcal{V}\backslash\mathcal{V}_L$ to ensure the attacks can be effective for various types of target nodes.

5 METHODOLOGY

In this section, we present the details of UGBA which aims to optimize Eq.(2) to conduct effective and unnoticeable graph backdoor attacks. Since it is challenging and computationally expensive to jointly optimize the selection of poisoned nodes $\mathcal{V}_P$ and the trigger generator, UGBA splits the optimization process into two steps: poisoned node selection and adaptive trigger generator learning. Two challenges remain to be addressed: (i) how to select the poisoned nodes that are most useful for backdoor attacks; (ii) how to learn the adaptive trigger generator to obtain triggers that meet unnoticeable constraint and maintain a high success rate in backdoor attack; To address these challenges, a novel framework of UGBA is proposed, which is illustrated in Fig. 2. UGBA is composed of a poisoned node selector f_P, an adaptive trigger generator f_g, and a surrogate GCN model f_s. Specifically, the poisoned node selector takes the graph $\mathcal{G}$ as input and applies a novel metric to select nodes with representative patterns in features and local structures as poisoned nodes. An adaptive trigger generator f_g is applied with a differentiable unnoticeable constraint to give unnoticeable triggers for selected poisoned nodes $\mathcal{V}_P$ to fool f_s. To guarantee the effectiveness of the generated adaptive triggers on various test nodes, a bi-level optimization with a surrogate GCN model is applied.

5.1 Poisoned Node Selection

In this subsection, we give the details of the node selection algorithm. Intuitively, if nodes with representative features and local

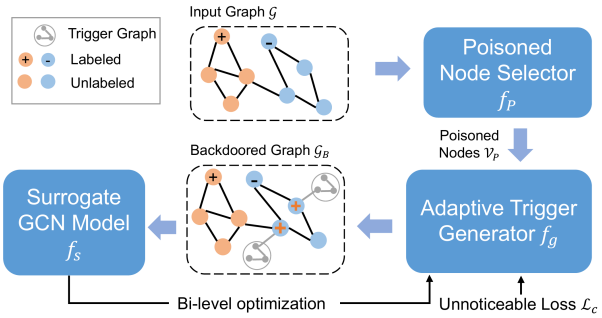


Figure 2: An overview of proposed UGBA.

structures are predicted to the target class y_t after being attached with triggers, other nodes are also very likely to be conducted successful backdoor attacks. Therefore, we propose to select diverse and representative nodes in the graph as poisoned nodes, which enforce the target GNN to predict the representative nodes attached with triggers to be target class y_t.

One straightforward way to obtain the representative nodes is to conduct clustering on the node features. However, it fails to consider the graph topology which is crucial for graph-structured data. Therefore, we propose to train a GCN encoder with the node labels to obtain representations that capture both attribute and structure information. Then, for each class, we can select representative nodes using the clustering algorithm on learned representations. Specifically, the node representations and labels can be obtained as:

$$\mathbf{H} = GCN(\mathbf{A}, \mathbf{X}), \quad \hat{\mathbf{Y}} = \text{softmax}(\mathbf{W} \cdot \mathbf{H}), \tag{3}$$

where $\mathbf{W}$ denotes the learnable weight matrix for classification. The training process of the GCN encoder can be written as:

$$\min_{\theta_E, \mathbf{W}} \sum_{v_i \in \mathcal{V}_L} l(\hat{y}_i, y_i) \tag{4}$$

where θ_E denotes the parameters of GCN encoder, $l(\cdot)$ is the cross entropy loss, and y_i is the label of node v_i. $\hat{y}_i$ is the prediction of v_i.

With the GCN encoder trained in Eq. (4), we can obtain the node representations and conduct clustering to obtain the representative nodes for each class. Here, to guarantee the diversity of the obtained representative nodes, we separately apply K-Means to cluster $\{\mathbf{h}_i : \hat{y}_i = l\}$ on each class l other than the target class y_t, where $\mathbf{h}_i$ denote the representation of node $v_i \in \mathcal{V}/\mathcal{V}_L$. Nodes nearer to the centroid of each cluster are more representative. However, the node nearest to the centroid may have a high degree. Injecting the malicious label to high-degree nodes may lead to a significant decrease in prediction performance as the negative effect will be propagated to its neighbors, which may make the attack noticeable. Hence, we propose a metric that balances the representativeness and negative effects on the prediction performance. Let $\mathbf{h}_c^k$ denote the center of the k-th cluster. Then for a node v_i^k belonging to the k-th cluster, the metric score can be computed by:

$$m(v_i) = ||\mathbf{h}_i^k - \mathbf{h}_c^k||_2 + \lambda \cdot deg(v_i^k) \tag{5}$$

where λ is to control the contribution of the degree in node selection. After getting each node's score, we select nodes with top-n highest scores in each cluster to satisfy the budget, where $n = \frac{\Delta_P}{(C-1)K}$.

5.2 Adaptive Trigger Generator

Once the poisoned node set $\mathcal{V}_P$ is determined, the next step is to generate adaptive triggers with f_g to poison the dataset. To guarantee the unnoticeability of the generated triggers, we propose a differentiable unnoticeable loss. We apply a bi-level optimization between the adaptive generator f_g and the surrogate model f_s to ensure high success rate on various test samples. Next, we give the details of trigger generator f_g, differentiable unnoticeable loss, and the bi-level optimization with f_s.

Design of Adaptive Trigger Generator. To generate adaptive triggers that are similar to the attached nodes, the adaptive trigger generator f_g takes the node features of the target node as input. Specifically, we adopt an MLP to simultaneously generate node features and structure of the trigger for node v_i by:

$$\mathbf{h}_i^m = \text{MLP}(\mathbf{x}_i), \quad \mathbf{X}_i^g = \mathbf{W}_f \cdot \mathbf{h}_i^m, \quad \mathbf{A}_i^g = \mathbf{W}_a \cdot \mathbf{h}_i^m, \tag{6}$$

where $\mathbf{x}_i$ is the node features of v_i. $\mathbf{W}_f$ and $\mathbf{W}_a$ are learnable parameters for feature and structure generation, respectively. $\mathbf{X}_i^g \in \mathbb{R}^{s \times d}$ is the synthetic features of the trigger nodes, where s and d represent the size of the generated trigger and the dimension of features, respectively. $\mathbf{A}_i^g \in \mathbb{R}^{s \times s}$ is the adjacency matrix of the generated trigger. As the real-world graph is generally discrete, following the binary neural network [16], we binarize the continuous adjacency matrix $\mathbf{A}_i^g$ in the forward computation; while the continuous value is used in backward propagation. With the generated trigger $g_i = (\mathbf{X}_i^g, \mathbf{A}_i^g)$, we link it to node $v_i \in \mathcal{V}_P$ and assign target class label y_t to build backdoored dataset. In the inference phase, the trigger generated by f_g will be attached to the test node $v_i \in \mathcal{V}_T$ to lead backdoored GNN to predict it as target class y_t.

Differentiable Unnoticeable Loss. The adaptive trigger generator f_g aims to produce the triggers that meet the Eq.(1) for unnoticeable trigger injection. The key idea is to ensure the poisoned node or test node v_i is connected to a trigger node with high cosine similarity to avoid trigger elimination. And within the generated trigger g_i, the connected trigger nodes should also exhibit high similarity. Thus, we design a differentiable unnoticeable loss to help optimize the adaptive trigger generator f_g. Let $\mathcal{E}_B^i$ denote the edge set that contains edges inside trigger g_i and edge attaching trigger g_i and node v_i, the unnoticeable loss can be written as:

$$\min_{\theta_g} \mathcal{L}_c = \sum_{v_i \in \mathcal{V}} \sum_{(v_j, v_k) \in \mathcal{E}_B^i} \max(0, T - sim(v_j, v_k)), \tag{7}$$

where T denotes the threshold of the similarity, and θ_g represents the parameters of f_g. The unnoticeable loss is applied on all nodes $\mathcal{V}$ to ensure that the generated trigger meets the unnoticeable constraint for various kinds of nodes.

Bi-level Optimization. To guarantee the effectiveness of the generated triggers, we optimize the adaptive trigger generator to successfully attack the surrogate GCN model f_s with a bi-level optimization. Specifically, the surrogate GCN f_s will be trained on the backdoored dataset, which can be formulated as:

$$\min_{\theta_s} \mathcal{L}_s(\theta_s, \theta_g) = \sum_{v_i \in \mathcal{V}_L} l(f_s(\mathcal{G}_C^i), y_i) + \sum_{v_i \in \mathcal{V}_P} l(f_s(a(\mathcal{G}_C^i, g_i)), y_t), \tag{8}$$

where θ_s represents the parameters of the surrogate GCN f_s, $\mathcal{G}_C^i$ indicates the clean computation graph of node v_i, and $a(\cdot)$ denotes

the attachment operation. y_i is the label of labeled node $v_i \in \mathcal{V}_L$ and y_t is the target class label. The adaptive trigger will be optimized to effectively mislead the surrogate model f_s to predict various nodes from $\mathcal{V}$ to be y_t once injected with adaptive triggers, which can be written as:

$$\mathcal{L}_g(\theta_s, \theta_g) = \sum_{v_i \in \mathcal{V}} l(f_s(a(\mathcal{G}_C^i, g_i)), y_t). \tag{9}$$

Combining the unnoticeable loss Eq.(7), the following bi-level optimization problem can be formulated:

$$\begin{aligned} &\min_{\theta_g} \mathcal{L}_g(\theta_s^*(\theta_g), \theta_g) + \beta \mathcal{L}_c(\theta_g) \\ &s.t. \quad \theta_s^* = \arg\min_{\theta_s} \mathcal{L}_s(\theta_s, \theta_g), \end{aligned} \tag{10}$$

where β is used to control the contribution of unnoticeable loss.

5.3 Optimization Algorithm

We propose an alternating optimization schema to solve the bi-level optimization problem of Eq.(10) with a small computation cost.

Updating Lower Level Surrogate Model. Computing θ_s^* for each outter iteration is expensive. We update surrogate model θ_s with N inner iterations with fixed θ_g to approximate θ_s^* as [46] does:

$$\theta_s^{t+1} = \theta_s^t - \alpha_s \nabla_{\theta_s} \mathcal{L}_s(\theta_s, \theta_g) \tag{11}$$

where θ_s^t denotes model parameters after t iterations. α_s is the learning rate for training the surrogate model.

Updating Upper Level Surrogate Model. In the outer iteration, the updated surrogate model parameters θ_s^T are used to approximate θ_s^*. Moreover, we apply first-order approximation [13] in computing gradients of θ_g to further reduce the computation cost:

$$\theta_g^{k+1} = \theta_g^k - \alpha_g \nabla_{\theta_g} \big(\mathcal{L}_g(\bar{\theta}_s, \theta_g^k) + \beta \mathcal{L}_c(\theta_g^k)\big), \tag{12}$$

where $\bar{\theta}_s$ indicates gradient propagation stopping. α_g is the learning rate of training adaptive generator. See more details in algorithm 1. And the time complexity analysis can be found in Appendix F.

6 EXPERIMENTS

In this section, we will evaluate proposed methods on various large-scale datasets to answer the following research questions:

- **RQ1**: Can our proposed method conduct effective backdoor attacks on GNNs and simultaneously ensure unnoticeability?
- **RQ2**: How do the number of poisoned nodes affect the performance of backdoor attacks?
- **RQ3**: How do the adaptive constraint and the poisoned node selection module affect the attack performance?

6.1 Experimental Settings

6.1.1 Datasets. To demonstrate the effectiveness of our UGBA, we conduct experiments on four public real-world datasets, i.e., Cora, Pubmed [23], Flickr [37], and OGB-arxiv [15], that are widely used for inductive semi-supervised node classification. Cora and Pubmed are small citation networks. Flickr is a large-scale graph that links image captions sharing the same properties. OGB-arixv is a large-scale citation network. The statistics of the datasets are summarized in Tab. 3.

Table 3: Dataset Statistics

Datasets	#Nodes	#Edges	#Feature	#Classes
Cora	2,708	5,429	1,443	7
Pubmed	19,717	44,338	500	3
Flickr	89,250	899,756	500	7
OGB-arxiv	169,343	1,166,243	128	40

6.1.2 Compared Methods. We compare UGBA with representative and state-of-the-art graph backdoor attack methods, including **GTA** [30], **SBA-Samp** [39] and its variant **SBA-Gen**. We also compare **GBAST** [24] on Pubmed, which is shown in the Appendix C.

As UGBA conduct attacks by injecting triggers to target nodes, we also compare UGBA with two state-of-the-art graph injection evasion attacks designed for large-scale attacks, i.e. **TDGIA** [44] and **AGIA** [5]. More details of these compared methods can be found in Appendix D. For a fair comparison, hyperparameters of all the attack methods are tuned based on the performance of the validation set.

Competing with Defense Methods. We applied the backdoor defense strategies introduced in Sec. 3.3.2 (i.e., Prune and Prune+LD) to help evaluate the unnoticeability of backdoor attacks. Moreover, two representative robust GNNs, i.e., **RobustGCN** [42] and **GNNGuard** [38], are also selected to verify that UGBA can also effectively attack general robust GNNs.

6.1.3 Evaluation Protocol. In this paper, we conduct experiments on the inductive node classification task, *where the attackers can not access test nodes when they poison the graph*. Hence, we randomly mask out 20% nodes from the original dataset. And half of the masked nodes are used as target nodes for attack performance evaluation. Another half is used as clean test nodes to evaluate the prediction accuracy of backdoored models on normal samples. The graph containing the rest 80% nodes will be used as training graph $\mathcal{G}$, where the labeled node set and validation set both contain 10% nodes. The average success rate (ASR) on the target node set and clean accuracy on clean test nodes are used to evaluate the backdoor attacks. A two-layer GCN is used as the surrogate model for all attack methods. And to demonstrate the transferability of the backdoor attacks, we attack target GNNs with different architectures, i.e., **GCN**, **GraphSage**, and **GAT**. Experiments on each target GNN architecture are conducted 5 times. We report the average ASR and clean accuracy of the total 15 runs (Tab. 4, Fig. 4, and Fig. 3). For all experiments, class 0 is the target class. The attack budget Δ_P on size of poisoned nodes $\mathcal{V}_P$ is set as 10, 40, 80 and 160 for Cora, Pubmed, Flickr and OGB-arxiv, respectively. The number of nodes in the trigger size is limited to 3 for all experiments. For experiments varying the budget in trigger size, please refer to Appendix E.

Our UGBA deploys a 2-layer GCN as the surrogate model. A 2-layer MLP is used as the adaptive trigger. More details of the hyperparameter setting can be found in Appendix B.

6.2 Attack Results

To answer **RQ1**, we compare UGBA with baselines on four real-world graphs under various defense settings in terms of attack performance and unnoticeability.

Table 4: Backdoor attack results (ASR (%) | Clean Accuracy (%)). Only clean accuracy is reported for clean graphs.

Datasets	Defense	Clean Graph	SBA-Samp	SBA-Gen	GTA	Ours
Cora	None	83.09	34.94 | 84.09	42.54 | 84.81	90.25 | 82.88	**96.95 | 83.90**
	Prune	79.68	16.70 | 82.98	19.56 | 83.19	17.63 | 83.06	**98.89 | 82.66**
	Prune+LD	79.68	15.87 | 79.63	17.49 | 80.61	18.35 | 80.17	**95.30 | 79.90**
Pubmed	None	84.86	30.43 | 84.93	31.96 | 84.93	86.64 | 85.07	**92.27 | 85.06**
	Prune	85.09	22.10 | 84.90	22.13 | 84.86	28.10 | 85.05	**92.87 | 85.09**
	Prune+LD	85.12	21.56 | 84.63	22.06 | 83.71	22.00 | 83.76	**93.06 | 83.75**
Flickr	None	46.40	0.00 | 47.36	0.00 | 47.07	88.64 | 45.67	**97.43 | 46.09**
	Prune	43.02	0.00 | 44.01	0.00 | 43.78	0.00 | 42.71	**90.34 | 42.99**
	Prune+LD	43.02	0.00 | 45.03	0.00 | 45.32	0.00 | 44.99	**96.81 | 42.14**
OGB-arxiv	None	65.50	0.65 | 65.53	11.26 | 65.43	75.01 | 65.54	**96.59 | 64.10**
	Prune	62.16	0.03 | 63.88	0.01 | 64.10	0.01 | 63.97	**93.07 | 62.58**
	Prune+LD	62.16	0.16 | 64.15	0.02 | 63.89	0.03 | 64.30	**90.95 | 63.19**

Table 5: Comparisons of ASR (%) with node inject attacks.

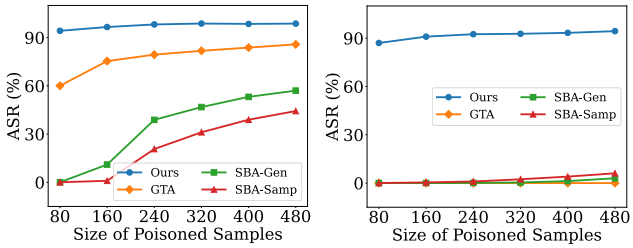
Datasets	Defense	TDGIA	AGIA	Ours
Flickr	GCN-Prune	77.01	77.22	**99.91**
	RobustGCN	78.61	78.61	**99.23**
	GNNGuard	55.68	56.01	**99.91**
OGB-arixv	GCN-Prune	66.17	66.33	**94.05**
	RobustGCN	73.87	74.00	**95.39**
	GNNGuard	42.27	42.58	**96.88**

6.2.1 Comparisons with baseline backdoor attacks. We conduct experiments on four real-world graphs under three backdoor defense strategy settings (i.e., No defense, Prune and Prune+LD). As described by the evaluation protocol in Sec. 6.1.3, we report the average results in backdooring three target GNN architectures in Tab. 4. The details of the backdoor attack results are presented in Tab. 8-10 in Appendix. From the table, we can make the following observations:

- When no backdoor defense strategy is applied, our UGBA outperforms the baseline methods, especially on large-scale datasets. This indicates the effectiveness of poisoned node selection algorithm in fully utilizing the attack budget.
- All the baselines give poor performance when the trigger detection based defense methods, i.e., Prune and Prune+LD, are adopted. By contrast, our UGBA can achieve over 90% ASR with the defense strategies and maintain high clean accuracy. This demonstrates that our UGBA can generate effective and unnoticeable triggers for backdoor attacks.
- As the ASRs are average results of backdooring three different GNN architectures, the high ASR scores of UGBA prove its transferability in backdooring various types of GNN models.

6.2.2 Comparisons with baseline node injection attacks. We also compare UGBA with two state-of-the-art node injection evasion attacks. Experiments are conducted on Flickr and OGB-arxiv. Three defense models (GCN-Prune, RobustGCN and GNNGuard) are selected to defend against the compared attacks. The ASR of 5 runs is reported in Tab 5. From this table, we observe:

- UGBA can effectively attack the robust GNNs, which shows that UGBA can also bypass the general defense methods with the unnoticeable constraint.



(a) No Defense **(b) Prune+LD**

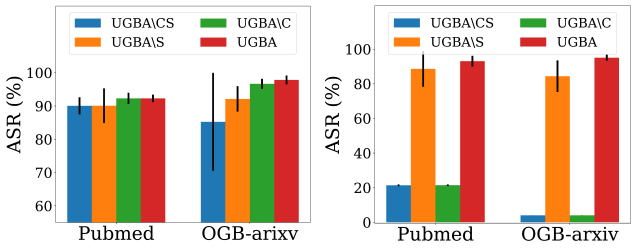
Figure 3: Impacts of sizes of poisoned nodes on OGB-arxiv.

- Compared with node injection attacks, UGBA only requires a very small additional cost in injecting triggers and labels (e.g. 160 poisoned nodes out of 169K nodes in OGB-arxiv). But UGBA can outperform node injection attacks by 30%. This implies the superiority of UGBA in attacking large amounts of target nodes.

6.3 Impacts of the Sizes of Poisoned Nodes

To answer **RQ2**, we conduct experiments to explore the attack performance of UGBA given different budgets in the size of poisoned nodes. Specifically, we vary the sizes of poisoned samples as $\{80, 160, 240, 320, 400, 480\}$. The other settings are the same as the evaluation protocol in Sec. 6.1.3. Hyperparameters are selected with the same process as described in Appendix. B. Fig. 4 shows the results on OGB-arxiv. We have similar observations on other datasets. From Fig. 4, we can observe that:

- The attack success rate of all compared methods in all settings increases as the increase of the number of poisoned samples, which satisfies our expectation. Our method consistently outperforms the baselines as the number of poisoned samples increases, which shows the effectiveness of the proposed framework. In particular, the gaps between our method and baselines become larger when the budget is smaller, which demonstrates the effectiveness of the poisoned node selection in effectively utilizing the attack budget.
- When Prune+LD defense is applied on the backdoor attacks, our methods still achieve promising performances, while all the baselines obtain nearly 0% ASR in all settings, which is as expected. That's because our method can generate trigger nodes similar to the attached nodes due to the unnoticeable constraint, which is helpful for bypassing the defense method.



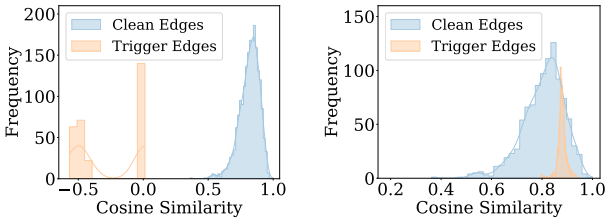
(a) No defense (b) Prune+LD

Figure 4: Ablation studies on Pubmed and OGB-arxiv.

6.4 Ablation Studies

To answer **RQ3**, we conduct ablation studies to explore the effects of the unnoticeable constraint and the poisoned node selection module. To demonstrate the effectiveness of the unnoticeable constraint module, we set the β as 0 when we train the trigger generator and obtain a variant named as UGBA\C. To show the benefits brought by our poisoned node selection module, we train a variant UGBA\S which randomly selects poisoned nodes to attach triggers and assign target nodes. We also implement a variant of our model by removing both unnoticeable constraint and poisoned node selection, which is named as UGBA\CS. The average results and standard deviations on Pubmed and OGB-arxiv are shown in Fig. 4. All the settings of evaluation follow the description in Sec. 6.1.3. And the hyperparameters of the variants are also tuned based on the validation set for fair comparison. From Fig. 4, we observe that:

- Compared with UGBA\S, UGBA achieves better attack results on various defense settings. The variance of ASR of UGBA is significantly lower than that of UGBA\S. This is because our poisoned node selection algorithm selects consistently diverse and representative nodes that are useful for backdoor attacks.
- When the backdoor defense strategy Prune+LD, UGBA can outperform UGBA\C and UGBA\CS by a large margin. This implies that the proposed unnoticeable loss manages to guide the trigger generator to give unnoticeable triggers for various test nodes, which can effectively bypass the pruning defenses.

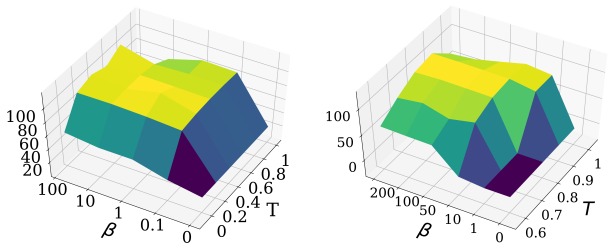


(a) GTA (b) UGBA

Figure 5: Edge similarity distributions on OGB-arxiv.

6.5 Similarity Analysis

In this section, we conduct a case study to further explore the similarity of the trigger nodes. We conduct backdoor attacks by using both GTA and our method on OGB-arxiv and then calculate the edge similarities of trigger edges (i.e., the edges associated with trigger nodes) and clean edges (i.e., the edges not connected to trigger nodes). The histogram of the edge similarity scores are plotted in Fig. 5. From the figure, we observe that the trigger edges generated by GTA have low similarities, which implies high risk of trigger elimination with our proposed backdoor defense strategies. In contrast, the edges created by our method present cosine similarity scores that well disguise them as clean edges, which verifies the unnoticeability of our methods.



(a) Pubmed (b) OGB-arxiv

Figure 6: Hyperparameter Sensitivity Analysis

6.6 Parameter Sensitivity Analysis

In this subsection, we further investigate how the hyperparameter β and T affect the performance of UGBA, where β and T control the weight of unnoticeable loss in training the trigger generator and the threshold of similarity scores used in unnoticeable loss. To explore the effects of β and T, we vary the values of β as $\{0, 50, 100, 150, 200\}$. And T is changed from $\{0, 0.2, 0.4, 0.6, 0.8, 1\}$ and $\{0.6, 0.7, 0.8, 0.9, 1\}$ for Pubmed and OGB-arxiv, respectively. Since β and T only affect the unnoticeablity of triggers, we report the attack success rate (ASR) of attacking against the Prune+LD defense strategy in Fig. 6. The test model is fixed as GCN. We observe that (i): In Pubmed, the similarity threshold T needs to be larger than 0.2; while T is required to be higher than 0.8 in OGB-arxiv. This is because edges in OGB-arxiv show higher similarity scores compared with Pubmed. Hence, to avoid being detected, a higher similarity threshold T is necessary. In practice, the T can be set according to the average edge similarity scores of the dataset. (ii) When T is set to a proper value, high ASR can generally be achieved when $\beta \leq 1$, which eases the hyperparameter tuning.

7 CONCLUSION AND FUTURE WORK

In this paper, we empirically verify that existing backdoor attacks require large attack budgets and can be easily defended with edge pruning strategies. To address these problems, we study a novel problem of conducting unnoticeable graph backdoor attacks with limited attack budgets. Specifically, a novel poisoned node selection algorithm is adopted to select representative and diverse nodes as poisoned nodes to fully utilize the attack budget. And an adaptive generator is optimized with an unnoticeable constraint loss to ensure the unnoticeability of generated triggers. The effectiveness of generated triggers is further guaranteed by bi-level optimization with the surrogate GCN model. Extensive experiments on large-scale datasets demonstrate that our proposed method can effectively backdoor various target GNN models and even be adopted with defense strategies. There are two directions that need further investigation. First, in this paper, we only focus on node classification. We will extend the proposed attack to other tasks such as recommendation and graph classification. Second, it is also interesting to investigate how to defend against the unnoticeable graph backdoor attack.

ACKNOWLEDGMENTS

This material is based upon work supported by, or in part by, the National Science Foundation (NSF) under grant number IIS-1707548 and IIS-1909702, the Army Research Office (ONR) under grant number W911NF21-1-0198, and Department of Homeland Security (DNS) CINA under grant number E205949D. The findings in this paper do not necessarily reflect the view of the funding agency.

REFERENCES

[1] Aleksandar Bojchevski and Stephan Günnemann. 2019. Adversarial Attacks on Node Embeddings via Graph Poisoning. In *Proceedings of the 36th International Conference on Machine Learning, ICML (Proceedings of Machine Learning Research)*. PMLR.

[2] Pietro Bongini, Monica Bianchini, and Franco Scarselli. 2021. Molecular generative Graph Neural Networks for Drug Discovery. *Neurocomputing* 450 (2021), 242–252.

[3] Heng Chang, Yu Rong, Tingyang Xu, Wenbing Huang, Honglei Zhang, Peng Cui, Wenwu Zhu, and Junzhou Huang. 2020. A Restricted Black-Box Adversarial Framework Towards Attacking Graph Embedding Models. *AAAI* 34, 3389–3396.

[4] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. 2020. Simple and deep graph convolutional networks. In *ICML*. 1725–1735.

[5] Yongqiang Chen, Han Yang, Yonggang Zhang, MA KAILI, Tongliang Liu, Bo Han, and James Cheng. 2022. Understanding and Improving Graph Injection Attack by Promoting Unnoticeability. In *ICLR*.

[6] Enyan Dai, Charu Aggarwal, and Suhang Wang. 2021. NRGNN: Learning a Label Noise-Resistant Graph Neural Network on Sparsely and Noisily Labeled Graphs. *arXiv preprint arXiv:2106.04714* (2021).

[7] Enyan Dai, Wei Jin, Hui Liu, and Suhang Wang. 2022. Towards robust graph neural networks for noisy graphs with sparse labels. In *WSDM*. 181–191.

[8] Enyan Dai and Suhang Wang. 2021. Say No to the Discrimination: Learning Fair Graph Neural Networks with Limited Sensitive Attribute Information. In *WSDM*. 680–688.

[9] Enyan Dai and Suhang Wang. 2021. Towards self-explainable graph neural network. In *CIKM*. 302–311.

[10] Enyan Dai, Tianxiang Zhao, Huaisheng Zhu, Junjie Xu, Zhimeng Guo, Hui Liu, Jiliang Tang, and Suhang Wang. 2022. A Comprehensive Survey on Trustworthy Graph Neural Networks: Privacy, Robustness, Fairness, and Explainability. *arXiv preprint arXiv:2204.08570* (2022).

[11] Enyan Dai, Shijie Zhou, Zhimeng Guo, and Suhang Wang. 2022. Label-Wise Graph Convolutional Network for Heterophilic Graphs. In *LOG*.

[12] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. 2018. Adversarial attack on graph structured data. *ICML* (2018).

[13] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*. 1126–1135.

[14] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*. 1024–1034.

[15] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. In *NeurIPS*, Vol. 33. 22118–22133.

[16] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. *NeurIPS*.

[17] John J Irwin, Teague Sterling, Michael M Mysinger, Erin S Bolstad, and Ryan G Coleman. 2012. ZINC: a free tool to discover chemistry for biology. *Journal of chemical information and modeling* 52, 7 (2012), 1757–1768.

[18] Dongkwan Kim and Alice Oh. 2021. How to find your friendly neighborhood: Graph attention design with self-supervision. In *ICLR*.

[19] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

[20] Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. In *ICLR*. 1885–1894.

[21] Yao Ma, Suhang Wang, Tyler Derr, Lingfei Wu, and Jiliang Tang. 2021. Graph Adversarial Attack via Rewiring. In *SIGKDD*. 1161–1169.

[22] Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. 2020. Gcc: Graph contrastive coding for graph neural network pre-training. In *SIGKDD*. 1150–1160.

[23] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI magazine* 29, 3 (2008), 93–93.

[24] Yu Sheng, Rong Chen, Guanyu Cai, and Li Kuang. 2021. Backdoor attack of graph neural networks based on subgraph trigger. In *International Conference on Collaborative Computing: Networking, Applications and Worksharing*. Springer, 276–296.

[25] Yiwei Sun, Suhang Wang, Xianfeng Tang, Tsung-Yu Hsieh, and Vasant Honavar. 2020. Adversarial Attacks on Graph Neural Networks via Node Injections: A Hierarchical Reinforcement Learning Approach. In *WWW*. Association for Computing Machinery, New York, NY, USA, 673–683.

[26] Shuchang Tao, Qi Cao, Huawei Shen, Junjie Huang, Yunfan Wu, and Xueqi Cheng. 2021. Single Node Injection Attack against Graph Neural Networks. In *CIKM*. 1794–1803.

[27] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. *ICLR* (2018).

[28] Binghui Wang, Tianxiang Zhou, Minhua Lin, Pan Zhou, Ang Li, Meng Pang, Cai Fu, Hai Li, and Yiran Chen. 2020. Evasion attacks to graph neural networks via influence function. *arXiv preprint arXiv:2009.00203* (2020).

[29] Daixin Wang, Jianbin Lin, Peng Cui, Quanhui Jia, Zhen Wang, Yanming Fang, Quan Yu, Jun Zhou, Shuang Yang, and Yuan Qi. 2019. A Semi-supervised Graph Attentive Network for Financial Fraud Detection. In *ICDM*. IEEE, 598–607.

[30] Zhaohan Xi, Ren Pang, Shouling Ji, and Ting Wang. 2021. Graph backdoor. In *USENIX Security*. 1523–1540.

[31] Junjie Xu, Enyan Dai, Xiang Zhang, and Suhang Wang. 2022. HP-GMN: Graph Memory Networks for Heterophilous Graphs. In *ICDM*. 1263–1268.

[32] Jing Xu and Stjepan Picek. 2022. Poster: Clean-label Backdoor Attack on Graph Neural Networks. In *CCS*. 3491–3493.

[33] Kaidi Xu, Hongge Chen, Sijia Liu, Pin-Yu Chen, Tsui-Wei Weng, Mingyi Hong, and Xue Lin. 2019. Topology Attack and Defense for Graph Neural Networks: An Optimization Perspective. In *IJCAI*. 3961–3967.

[34] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).

[35] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *SIGKDD*. 974–983.

[36] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. 2019. Gnnexplainer: Generating explanations for graph neural networks. In *Advances in neural information processing systems*. 9244–9255.

[37] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2020. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *ICLR*.

[38] Xiang Zhang and Marinka Zitnik. 2020. GNNGuard: Defending Graph Neural Networks against Adversarial Attacks. In *NeurIPS*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 9263–9275.

[39] Zaixi Zhang, Jinyuan Jia, Binghui Wang, and Neil Zhenqiang Gong. 2021. Backdoor attacks to graph neural networks. In *SACMAT*. 15–26.

[40] Zaixi Zhang, Qi Liu, Hao Wang, Chengqiang Lu, and Cheekong Lee. 2022. Protgnn: Towards self-explaining graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 9127–9135.

[41] Tianxiang Zhao, Xianfeng Tang, Xiang Zhang, and Suhang Wang. 2020. Semi-Supervised Graph-to-Graph Translation. In *CIKM*. 1863–1872.

[42] Dingyuan Zhu, Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2019. Robust graph convolutional networks against adversarial attacks. In *SIGKDD*. 1399–1407.

[43] Qikui Zhu, Bo Du, and Pingkun Yan. 2020. Self-supervised Training of Graph Convolutional Networks. *arXiv preprint arXiv:2006.02380* (2020).

[44] Xu Zou, Qinkai Zheng, Yuxiao Dong, Xinyu Guan, Evgeny Kharlamov, Jialiang Lu, and Jie Tang. 2021. TDGIA: Effective Injection Attacks on Graph Neural Networks. In *SIGKDD*. Association for Computing Machinery, New York, NY, USA, 2461–2471.

[45] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. 2018. Adversarial attacks on neural networks for graph data. In *SIGKDD*. 2847–2856.

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

Algorithm 1 Algorithm of UGBA.

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, $\mathcal{Y}_L$, β, T.
Output: Backdoored dataset $\mathcal{G}_B$, adaptive trigger generator f_g

1: Initialize $\mathcal{G}_B = \mathcal{G}$;
2: Randomly initialize θ_s and θ_g for f_s and f_g;
3: Select poisoned nodes $\mathcal{V}_P$ based on Eq. (5);
4: Assign class t as labels of $\mathcal{V}_P$;
5: **while** not converged yet **do**
6: **for** t=1, 2, . . . , N **do**
7: Update θ_s by descent on $\nabla_{\theta_s}\mathcal{L}_s$ based on Eq. (11);
8: **end for**
9: Update θ_g by descent on $\nabla_{\theta_g}\left(\mathcal{L}_g + \beta\mathcal{L}_c\right)$ based on Eq. (12);
10: **end while**
11: **for** $v_i \in \mathcal{V}_P$ **do**
12: Generate the trigger g_i for v_i by using f_g;
13: Update $\mathcal{G}_B$ based on $a(\mathcal{G}_B^i, g_i)$;
14: **end for**
15: **return** $\mathcal{G}_B$ and f_g;

A TRAINING ALGORITHM

The algorithm of UGBA is proposed in Algorithm 1. Specially, we first select the poisoned nodes $\mathcal{V}_P$ with the top-n highest scores $m(\cdot)$ based on Eq. (5), and assign the target class y_t as labels to $\mathcal{V}_P$ (lines 3-4). From line 5 to line 10, we train the trigger generator f_g on the surrogate GCN f_s by solving a bi-level optimization problem based on Eq. (10). In detail, we update lower level surrogate model (lines 6-8) and upper level surrogate model (line 9), respectively, by doing gradient descent on θ_s and θ_g based on Eq. (11) and Eq. (12). After that, from line 11 to line 14, we use the well-trained f_g to generate a trigger g_i for each poisoned node $v_i \in \mathcal{V}_P$ and attach g_i with v_i to obtain the poisoned graph $\mathcal{G}_B$.

B IMPLEMENTATION DETAILS

A 2-layer GCN is deployed as the surrogate model. A 2-layer MLP is used as the adaptive trigger. All the hidden dimension is set as 32. The inner iterations step N is set as 5 for all the experiments. For the hyperparameter β and T, they are selected based on the grid search on the validation set. Specifically, T is fixed as 0.5, 0.5, 0.5, 0.8 for Cora, Pubmed, Flickr and OGB-arxiv, respectively. For Prune and Prune+LD defenses, the threshold of pruning is set to filter out around 10% dissimilar edges. In particular, the set thresholds are around 0.1, 0.2, 0.4, 0.8 for Cora, Pubmed, Flickr and OGB-arxiv.

C ADDITIONAL EXPERIMENTS

We compare our UGBA with GBAST [24] on Pubmed. And we report the ASR (%) of attacking GCN under different defense settings in Tab. 6. We similar observations on other datasets and target GNN models. Compared with GBAST which selects the poisoned samples by degree and closeness centrality, our UGBA achieves much higher ASR when not defense is applied. This implies the effectiveness of our clustering-based poisoned sample selection. In addition, GBAST cannot bypass the defense strategy. Our UGBA can still show high attack performance under the Prune+LD defense.

Table 6: Comparison with GBAST

Defense	GBAST	UGBA
None	55.1 ±11.4	96.3 ±1.3
Prune+LD	21.3 ±0.2	92.3 ±2.1

D DETAILS OF COMPARED METHODS

The details of compared methods are described following:

- **SBA-Samp** [39]: It injects one fixed subgraph as a trigger to the training graph for a poisoned node. To generate the subgraph the connections are generated using Erdos-Renyi (ER) model and the node features are randomly sampled from the training graph.
- **SBA-Gen**: This is a variant of SBA-Samp, which uses generated features for trigger nodes. Features are from a Gaussian distribution whose mean and variance is computed from real nodes.
- **GTA** [30]: This is the state-of-the-art backdoor attack on GNNs. Poisoned nodes is randomly selected in GTA. A trigger generator is adopted to create subgraphs as sample-specific triggers. The trigger generator is purely optimized by the backdoor attack loss without any unnoticeable constraint.
- **TDGIA** [44]: It employs a topological defective edge selection strategy to choose the nodes to be connecting with the injected ones, and generates the features for injected nodes by performing the smooth adversarial feature optimization.
- **AGIA** [5]: It leverages gradient information to perform a bi-level optimization for the features and structures of the injected nodes.

E IMPACTS OF TRIGGER SIZE

In this section, we conduct experiments to explore the attack performance of UGBA by injecting different numbers of nodes as a trigger for a poisoned node. Specially, the trigger size is varied as $\{1, 2, 3, 4, 5\}$. The other settings are the same as the evaluation protocol in Sec. 6.1.3. The results on OGB-arxiv are shown in Table 7. We have similar observations on other datasets. From the table, we can find that: (i) as the increase of trigger sizes, the attack success rate of UGBA in all settings increases, as larger trigger can be stronger in backdoor attack; (ii) UGBA can achieve stable and high attack performance when the trigger size is as small as 2, which shows the effectiveness of our UGBA in generating triggers.

Table 7: Attack results of UGBA (ASR (%)) with various trigger sizes under three defense strategies on OGB-arxiv

Trigger Size	1	2	3	4	5
None	83.0	98.9	98.8	98.9	97.8
Prune	77.7	94.5	94.4	94.6	93.8
Prune+LD	65.2	94.0	95.1	94.2	89.0

F TIME COMPLEXITY ANALYSIS

In the poisoning phase, the time complexity mainly comes from the poisoned node selection and the optimization of trigger generator. Let h denote the embedding dimension. The cost of poisoned node selection with clustering is approximately $O(Kh|\mathcal{V}|)$, where K is the number of clusters set in poisoned node selection and $|\mathcal{V}|$ is the number of nodes in the training graph. During the bi-level optimization phase, the computation cost of each outter iteration

Table 8: Results of backdooring GCN (ASR (%) | Clean Accuracy (%)). Only clean accuracy is reported for clean graph.

Datasets	Defense	Clean Graph	SBA-Samp	SBA-Gen	GTA	Ours
Cora	None	82.9	33.8±3.4 | 83.9±1.1	40.1±7.0 | 83.5±1.1	98.9±0.8 | 82.7±1.2	**98.8±0.1** | **83.5±0.8**
	Prune	79.6	17.1±2.3 | 83.1±1.1	19.9±2.6 | 83.3±0.9	17.7±3.2 | 83.6±0.6	**99.6±0.0** | **82.5±0.9**
	Prune+LD	79.6	16.6±2.7 | 81.3±1.2	18.9±3.5 | 81.3±1.0	20.1±5.6 | 80.8±0.4	**99.6±0.1** | **81.5±0.6**
Pubmed	None	85.1	26.4±2.9 | 84.9±0.2	28.8±3.5 | 85.0±0.2	92.8±2.9 | 85.2±0.2	**96.3±1.2** | **84.9±0.1**
	Prune	85.1	22.4±1.0 | 85.3±0.1	22.6±0.9 | 85.2±0.1	28.8±1.2 | 85.1±1.0	**93.0±1.1** | **85.4±0.2**
	Prune+LD	85.1	21.7±0.9 | 85.0±0.2	22.2±1.2 | 84.2±0.3	22.3±0.5 | 84.2±0.1	**92.3±2.0** | **85.0±0.1**
Flickr	None	45.5	0±0.0 | 46.7±0.3	0±0.0 | 46.4±0.1	99.9±0.1 | 45.0±0.3	**96.9±2.3** | **44.8±0.4**
	Prune	42.3	0±0.0 | 44.1±0.2	0±0.0 | 43.4±0.4	0±0.0| 41.7±0.2	**99.9±0.0** | **41.7±0.4**
	Prune+LD	42.3	0±0.0 | 45.6±0.2	0±0.0 | 45.6±0.2	0±0.0 | 44.5±0.4	**96.6±1.6** | **44.8±0.1**
OGB-arxiv	None	65.6	0.3±0.1 | 65.8±0.1	1.8±2.4 | 65.8±0.2	75.2±1.4 | 65.8±0.1	**98.8±0.1** | **63.9±0.5**
	Prune	62.1	0.1±0.1 | 64.5±0.5	0.1±0.1 | 64.0±0.1	0.1±0.1 | 64.0±0.1	**94.0±0.3** | **62.2±0.7**
	Prune+LD	62.1	0.1±0.1 | 64.7±0.1	0.1±0.1 | 64.6±0.1	0.1±0.1 | 64.7±0.2	**93.5±0.2** | **63.0±0.4**

Table 9: Results of backdooring GraphSage (ASR (%) | Clean Accuracy (%)). Only clean accuracy is reported for clean graph.

Datasets	Defense	Clean Graph	SBA-Samp	SBA-Gen	GTA	Ours
Cora	None	81.8	34.2±4.0 | 83.0±1.5	40.4±5.6 | 82.7±1.2	99.5±0.4 | 81.3±1.0	**92.7±2.1** | **82.8±1.4**
	Prune	77.9	17.3±1.8 | 82.5±1.1	19.9±3.6 | 82.4±1.1	18.6±4.1 | 81.9±0.8	**99.6±0.1** | **83.9±1.5**
	Prune+LD	77.9	16.4±1.8 | 78.8±0.6	16.9±3.2 | 78.9±0.8	18.0±4.5 | 77.9±2.0	**94.4±2.1** | **77.8±0.5**
Pubmed	None	85.7	38.0±3.8 | 85.8±0.3	40.0±4.2 | 85.9±0.2	92.7±3.5 | 86.0±0.3	**96.0±0.9** | **86.0±0.1**
	Prune	86.2	22.8±1.0 | 85.8±0.2	23.0±0.9 | 85.8±0.1	27.4±1.2 | 86.5±0.3	**91.0±0.6** | **86.4±0.1**
	Prune+LD	86.2	21.5±1.0 | 85.4±0.2	22.0±1.2 | 83.8±0.1	21.9±0.3 | 83.8±0.2	**91.6±1.7** | **86.0±0.1**
Flickr	None	47.0	0±0.0 | 48.5±0.1	0±0.0 | 48.4±0.1	99.7±0.2 | 48.0±0.3	**98.9±0.3** | **47.7±0.1**
	Prune	45.2	0±0.0 | 46.7±0.1	0±0.0 | 46.8±0.1	0±0.0 | 45.9±0.2	**98.9±0.9** | **41.2±1.3**
	Prune+LD	45.2	0±0.0 | 44.4±0.4	0±0.0 | 44.4±0.4	0±0.0 | 44.5±0.4	**97.1±2.4** | **44.7±0.3**
OGB-arxiv	None	65.6	0.5±0.6 | 65.4±0.6	6.2±3.5 | 65.3±0.6	55.5±2.3 | 65.9±0.3	**91.0±0.8** | **63.6±0.6**
	Prune	62.5	0.1±0.1 | 64.5±0.4	0.1±0.1 | 64.5±0.5	0.1±0.1 | 64.8±0.4	**89.7±0.6** | **62.6±0.4**
	Prune+LD	62.5	0.3±0.4 | 63.7±0.5	0.1±0.1 | 63.8±0.5	0.1±0.1 | 64.1±0.3	**84.6±0.5** | **62.8±0.2**

Table 10: Results of backdooring GAT (ASR (%) | Clean Accuracy (%)) . Only clean accuracy is reported for clean graph.

Datasets	Defense	Clean Graph	SBA-Samp	SBA-Gen	GTA	Ours
Cora	None	84.5	36.8±8.7 | 85.4±1.3	47.1±18.0 | 84.4±1.1	72.3±27.7 | 84.6±0.8	**99.3±0.7** | **85.4±1.0**
	Prune	81.2	15.7±2.4 | 83.7±0.9	18.9±3.6 | 83.8±0.6	16.6±1.5 | 83.7±1.3	**99.6±0.1** | **83.9±1.5**
	Prune+LD	81.2	14.6±2.9 | 81.1±1.2	16.7±3.8 | 81.6±0.8	16.9±4.4 | 81.8±1.2	**92.0±14.7** | **80.4±0.8**
Pubmed	None	83.9	26.9±4.5 | 84.1±0.3	27.1±3.8 | 83.9±0.2	91.2±1.5 | 84.0±0.2	**100±0.0** | **84.0±4.2**
	Prune	84.0	21.1±0.9 | 83.6±0.2	20.8±1.4 | 83.6±0.2	28.1±1.1 | 83.5±0.1	**94.6±2.6** | **83.5±0.2**
	Prune+LD	84.0	21.5±1.3 | 83.5±0.3	22.0±0.8 | 83.2±0.3	21.8±0.4 | 83.3±0.5	**95.3±4.1** | **84.0±4.2**
Flickr	None	46.5	0±0.0 | 46.9±0.2	0±0.0 | 46.4±0.4	66.2±34.9 | 44.0±0.6	**96.5±4.4** | **45.8±1.3**
	Prune	41.7	0±0.0 | 41.2±0.8	0±0.0 | 41.2±1.2	0±0.0 | 40.5±0.1	**72.2±27.5** | **41.2±1.3**
	Prune+LD	41.7	0±0.0 | 46.1±0.6	0±0.0 | 46.0±0.6	0±0.0 | 45.7±0.4	**96.7±6.3** | **46.0±0.3**
OGB-arxiv	None	65.3	1.1±1.3 | 65.4±0.1	25.8±20.3 | 65.2±0.6	94.3±2.5 | 65.0±0.1	**100±0.0** | **64.8±0.2**
	Prune	61.9	0.1±0.1 | 63.2±0.1	0.1±0.1 | 63.2±0.1	0.1±0.1 | 63.1±0.1	**95.5±0.0** | **62.9±0.2**
	Prune+LD	61.9	0.1±0.1 | 64.1±0.1	0.1±0.1 | 64.0±0.1	0.1±0.1 | 64.1±0.1	**94.8±0.0** | **63.7±0.1**

Table 11: Training time

Dataset	Size	GTA	UBGA
Flickr	89,250	18.1s	18.3s
Arxiv	169,343	37.7s	41.8s

We also report the overall training time of our UGBA and GTA in Tab. 11. All models are trained with 200 epochs on an A6000 GPU with 48G memory. The training time is very short and increases linearly as the complexity analysis suggests. In the test phase, attacking each target node requires 0.0017 seconds on average.

consists of updating of surrogate GCN model in inner iterations and training adaptive trigger generator. The cost for updating surrogate model is approximate $O(Nhd|\mathcal{V}|)$, where d is the average degree of nodes and N is the number of inner iterations which is generally small. The cost for optimizing the trigger generator in each outter iteration is $O(hd|\mathcal{V}|)$. Hence, the overall time complexity in each iteration of optimization is $O((N+1)hd|\mathcal{V}|)$, which is linear to the graph size. Furthermore, the framework can be trained in a mini-batch way to further reduce the computation cost in each iteration. In the test phase, the cost of generating trigger to attack the target node is only $O(h)$. Our time complexity analysis proves that UGBA has great potential in conducting scalable target attacks.