Curriculum Graph Poisoning

Hanwen Liu*†
Peking University
Beijing, China
hanwenliu@msn.com

Peilin Zhao*
Tencent AI Lab
Shenzhen, China
masonzhao@tencent.com

Tingyang Xu
Tencent AI Lab
Shenzhen, China
tingyangxu@tencent.com

Yatao Bian
Tencent AI Lab
Shenzhen, China
yatao.bian@gmail.com

Junzhou Huang
University of Texas at Arlington
Arlington, United States
jzhuang@uta.edu

Yuesheng Zhu
Peking University
Shenzhen, China
zhuys@pku.edu.cn

Yadong Mu‡
Peking University
Beijing, China
myd@pku.edu.cn

ABSTRACT

Despite the success of graph neural networks (GNNs) over the Web in recent years, the typical transductive learning setting for node classification requires GNNs to be retrained frequently, making them vulnerable to poisoning attacks by corrupting the training graph. Poisoning attacks on graphs are, however, non-trivial as the attack space is potentially large, and the discrete graph structure makes the poisoning function non-differentiable. In this paper, we revisit the bi-level optimization problem in graph poisoning and propose a novel graph poisoning method, termed **Cu**rriculum **G**raph **Po**isoning (`CuGPo`), inspired by curriculum learning. In contrast to other poisoning attacks that use heuristics or directly optimize the graph, our method learns to generate poisoned graphs from basic adversarial knowledge first and advanced knowledge later. Specifically, for the outer optimization, we utilize the slightly perturbed graphs which represent the easy poisoning task at the beginning, and then enlarge the attack space until the final; for the inner optimization, we firstly exploit the knowledge from the clean graph and then adapt quickly to perturbed graphs to obtain the adversarial knowledge. Extensive experiments demonstrate that `CuGPo` achieves state-of-the-art performance in graph poisoning attacks.

CCS CONCEPTS

• Security and privacy; **• Computing methodologies → Neural networks**; **• Mathematics of computing → Graph algorithms**;

KEYWORDS

graph neural networks, poisoning attacks, curriculum learning

ACM Reference Format:
Hanwen Liu, Peilin Zhao, Tingyang Xu, Yatao Bian, Junzhou Huang, Yuesheng Zhu, and Yadong Mu. 2023. Curriculum Graph Poisoning. In *Proceedings of the ACM Web Conference 2023 (WWW '23), April 30-May 4, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3543507.3583211

*Equal contributions.
†Work done during the internship at Tencent AI Lab.
‡Corresponding author.

1 INTRODUCTION

Graph neural networks (GNNs) have achieved significant success in a variety of important applications involving relational information from semi-supervised node classification [2, 31, 63, 71] to graph classification [35, 64, 65]. As GNNs have already been deployed in many real-world applications, especially over the Web [8, 33], in recent years, it is particularly concerning that they are shown to be vulnerable to adversarial attacks via carefully crafted perturbations [61, 77], including poisoning attacks and evasion attacks.

To mislead the model, poisoning attacks have been extensively investigated in the context of adversarial machine learning [5, 18]. Unlike evasion attacks that manipulate input data (*i.e.*, adversarial examples) at test time [23], poisoning attacks aim to inject specially crafted data points into the training data to worsen test accuracy [17, 28, 44, 46, 50, 67] when the test data or queries are not controlled by the adversary [20, 56]. Considering that node classification is typically performed in the setting of transductive learning, where both the training data and the test data are used jointly to learn the parameters of GNNs, under this setting it is natural to frequently retrain GNN models to get the latest relationship representations, making it vital to understand the vulnerability of GNNs in the poisoning attack domain. Pioneering graph poisoning attacks [76, 78] focus on manipulating the existing graph structure, *i.e.*, inserting and deleting edges between existing nodes. Unfortunately, in many real-world Web scenarios (*e.g.*, citation networks) modifying the existing graphs is infeasible, as these graphs are already stored in the database. However, since the transductive setting demands frequent updates, injecting fake nodes (*e.g.*, publishing fake manuscripts) with connections to existing nodes is far more achievable. Therefore, recent studies [49, 75] concentrate on another attack, namely the graph injection attack, by injecting new nodes with malicious intentions. In this work, we mainly focus on graph poisoning attacks via malicious node injections.

Learning a bad GNN model by polluting the training graph is non-trivial. Essentially, the poisoning attack can be viewed as a game where the victim and the adversary compete with each other: the victim tries to train a good model as usual, while the adversary aims to corrupt the training graph without triggering an alert. As is empirically proved, simply injecting a few nodes without new edges, or randomly perturbing a few edges of the training graph cannot significantly reduce the GNN performance. Besides, on account of the discreteness of graph structure and the potentially large attack space, neither directly optimizing the poisoned graph using the gradient-based method, nor the brute-force search is feasible.

Though there exists a few graph poisoning methods, challenges remain. On one hand, the prior graph poisoning method [78] by optimizing the graph [19] is extremely memory-inefficient at scale, since it needs to optimize the entire training graph recurrently. On the other hand, the previous node injection method [49] based on deep Q networks [26] suffers from slow and occasionally unstable convergence in reality. These pitfalls motivate us to think about how to efficiently and effectively learn a good poisoning policy. To this end, we explore the concept of difficulty in graph poisoning from the adversary's perspective and find out that the key to such limitations is learning a simple one first.

Contributions. In this paper, we propose a novel graph poisoning method called **Cu**rriculum **G**raph **Po**isoning (CuGPo). Our essential insight is that the poisoning method can learn trivial yet easier graph poisoning tasks at the beginning and then fully exploit the learned adversarial knowledge to tackle harder ones, in a fashion of curriculum learning [3]. To create curricula, we revisit the bi-level optimization problem in graph poisoning attacks: a) for the outer optimization which attempts to maximize the adversarial loss by optimizing the poisoned graph, CuGPo is introduced to the poisoning concepts by progressively increasing task complexity. b) for the inner optimization which aims to minimize the training loss on the poisoned graph, CuGPo learns to generalize on the clean training graph before adapting to various poisoned graphs at different task levels, to boost up and accelerate the inner optimization process. In particular, our proposed CuGPo is not only more efficient but also outperforms previous state-of-the-art graph poisoning methods across different victim models and different graphs, with detailed analysis and empirical demonstrations.

2 RELATED WORK

There has been a growing trend towards adversarial learning in graph neural networks, including evasion attacks in graph learning [16, 21, 38, 39, 54] and poisoning attacks against graph-based semi-supervised learning [34]. In poisoning attacks, there exists the bi-level optimization problem, as the adversary dedicates to corrupting the training data to degrade the predictive performance after training. Compared with *Euclidean data*, poisoning attacks against graphs [69] are generally more complex due to discreteness [37, 74]. Earlier work [76] tried to steer clear of the bi-level optimization problem by attacking only a single node based on a static surrogate model. [78] proposed to tackle this problem by treating the input graph as a hyper-parameter and optimizing it using meta-gradient descent, and also discussed a simple heuristic [57]. [32] proposed to boost the gradient-based adversarial perturbations on graphs by using an exploratory strategy, including phases of generation, evaluation, and recombination. [49] proposed a method based on reinforcement learning for graph injection attacks that injects malicious nodes into the original graph [55, 75]. Also, some works focus on vulnerabilities in other tasks, including node embedding [7], graph matching [70], graph label-flipping [68], graph backdoor attacks [60], and graph out-of-distribution problems [59].

Curriculum learning [3, 24, 48, 66, 72] has been investigated to improve the performance of deep learning systems, *e.g.*, recommender systems [11]. In this work, we mainly discuss the insight of curriculum learning for tackling the bi-level optimization in graph poisoning. Unlike previous curriculum learning studies that focus on *improving* the accuracy, we explore the opposite direction and design curricula from the adversary's view to *worsen* the accuracy under certain constraints.

3 CURRICULUM GRAPH POISONING

In this part, we formally define graph poisoning attacks and discuss the valid poisoning policy, and lastly introduce the proposed CuGPo.

3.1 Deep Graph Learning

As graph learning in a transductive setting is inherently related to poisoning attacks [77], we mainly focus on semi-supervised node classification tasks in this paper. Given a graph $G = \{A, X\}$ with the adjacency matrix $A = \{a_{uv} \mid u, v \in V\}$ and the node feature matrix $X = \{x_u \mid u \in V\}$ where V denotes the node set, we can train a graph neural network f_ω with the parameters ω by optimizing a classification loss ℓ_{train} (*e.g.*, the negative log-likelihood loss). For graph convolutional networks [30], f_ω learns the graph representations by layer-wise aggregating messages:

$$H^{(l+1)} \leftarrow \mathrm{ReLU}(\hat{A} H^{(l)} \omega^{(l)}), \tag{1}$$

where $\omega^{(l)}$ denotes the weight matrix at layer l of f_ω, and $H^{(l+1)}$ refers to the output hidden representation, starting with $H^0 = X$. $\hat{A}$ represents the (transformed) symmetric adjacency matrix *w.r.t.* A. In the transductive learning setting, nodes in the set of labelled nodes $V_L \subseteq V$ with the corresponding label set $Y = \{y_u \mid u \in V_L\}$, along with the nodes (but not their labels) in the unlabelled node set $V_U = V \setminus V_L$ are available while training, and the goal is to infer labels of nodes in V_U.

3.2 Graph Poisoning Attacks

Threat model. We investigate an adversary named *Eve*[1] who has the same knowledge about the training graph as the victim and intends to poison part of the training graph. Considering the most challenging setting, *Eve* has no prior knowledge about the target classifier, including its parameters or architecture. With the aim of degrading the test performance after model training, *Eve* tries to modify the training graph with minimum changes. That is to say, *Eve* is supposed to learn a poisoning policy function π_θ with the parameters θ to transform the benign training graph G into the poisoned graph $G' = \{A', X'\}$ with the budget Δ (*i.e.*, constraints on the adversarial capability of *Eve*), to increase test loss ℓ_{test}. The budget Δ is designed to make changes unnoticeable. However, unlike the image domain where we can constrain the unnoticeable property using human supervision under numerical constraints, it is hard to define such unnoticeability in the graph domain. Therefore, we consider a cardinality constraint $\Delta(G) = (\Delta_A, \Delta_X) \in \mathbb{Z} \times \mathbb{Z}$ *w.r.t.* the benign graph G to define a valid poisoning policy, and other options about unnoticeability are discussed in Section 3.3.

[1]We use *Eve* to denote the attacker as it is a common name in cryptography [4].

As in many graph-based applications, injecting new nodes and edges can be much more feasible than manipulating the existing graphs in the database, thus we mainly focus on graph poisoning attacks based on malicious node injections aligned with Δ. Particularly, assuming the malicious node set for graph injection attacks to be V_I, we have the adjacency matrix $A' = \{a_{uv} \mid u, v \in V_I \cup V\}$ and the node feature matrix $X' = \{x_u \mid u \in V_I \cup V\}$ after graph injection attacks. From the perspective of *Eve*, the primary goal is to find a valid graph poisoning policy to generate A' and X' under a constrained budget. We follow the common setting [29] for poisoned features X' and leave them out of the optimization scope. In other words, the poisoned features X' (along with the isolated V_I) are pre-defined ahead, making the poisoning procedure towards maliciously inserting edges between the injected nodes and the partially poisoned graph. Formally, the definition of a valid poisoning policy discussed in this paper is generalized as:

Definition 3.1 (Valid poisoning policy). Assume graph injection attacks are concerned under a cardinality constraint $\Delta(G) = (\Delta_A, \Delta_X)$ that depends on G. Given a benign graph $G = \{A, X\}$, a target victim model f_ω and the pre-defined $X' = \{x_u \mid u \in V_I \cup V\}$, we define a (G, f_ω)-valid poisoning policy function π_θ as, $\forall G' = \{A', X'\}$ where $A' \in \{A' \mid ||A' - A||_0 \leq \Delta_A\}$ and $|V_I| \leq \Delta_X$, there exists $G^* = \{A^*, X'\} \sim \mathbb{P}_{\pi_\theta}(G \mid f_\omega)$ that satisfies $\ell_{test}(f_{\omega^*}(G')) \leq \ell_{test}(f_{\omega^*}(G^*))$ and $A^* \in \{A' \mid ||A' - A||_0 \leq \Delta_A\}$.

Bi-level optimization. From an information propagation perspective, poisoning attacks are more challenging than evasion attacks on account of the typically non-convex training procedure [36], as the goal is to worsen the test performance after poisoning the data and the training process. As is described in Definition 3.1, for a valid poisoning policy π_θ, the objective function for non-targeted poisoning attacks can be formulated as a bi-level optimization problem:

$$\begin{aligned} \max_{G' \sim \mathbb{P}_{\pi_\theta}(G|f_\omega)} \quad & \ell_{test}(f_{\omega^*}(G')) \\ \text{s.t.} \quad & \omega^* = \arg\min_{\omega} \ell_{train}(f_\omega(G')) \\ & ||A' - A||_0 \leq \Delta_A. \end{aligned} \tag{2}$$

Since labels of the test data are not accessible to *Eve*, we use a surrogate GNN to estimate the generalization loss (*i.e.*, taking the pseudo-labels estimated by the surrogate model as the ground-truth labels of the test data). Also, we use another surrogate model as f_ω to evaluate the attack performance under the assumption that *Eve* has no knowledge about the target victim classifier.

As the transductive learning setting indicates that the training graph may change frequently, *Eve* has to find a poisoning policy that adapts to the changed graph. Therefore, merely a valid poisoning policy without strong generalization ability is not enough: a) in the graph domain, it is extremely time-consuming to solve this bi-level optimization problem multiple times; b) the more attempts *Eve* makes, which will alert the security department of the target graph-based system, the fewer chances (*e.g.*, for obtaining the training graph) *Eve* has. To overcome such limitations, in this paper we first propose the problem of graph-agnostic poisoning attacks defined as follows, and later empirically prove that our proposed CuGPo is graph-agnostic, which only needs to be trained once to perform poisoning attacks on different graphs for a given graph distribution.

Definition 3.2 ($\mathcal{G}$-agnostic poisoning policy). Given a graph distribution $\mathcal{G}$, a victim model f_ω and the budget Δ, a $\mathcal{G}$-agnostic poisoning policy π_θ is defined as, $\forall G \sim \mathbb{P}_{\mathcal{G}}(G)$, π_θ satisfies Eq. (2) with an optimal solution (*i.e.*, G^*).

3.3 A Markov Decision Process Perspective

Since the concept of difficulty in graph poisoning is important to the curricula design, we first use a simple policy, namely the brute-force search policy, to investigate the poisoning complexity. According to the threat model in Section 3.2, given a benign graph G and a budget Δ as a cardinality constraint, we can always find the optimal poisoned graph that achieves the worst test accuracy for a victim model f_ω (Proposition 3.3). However, to find the optimal poisoned graph via the brute-force search policy, the complexity is bounded as $O((\frac{e|V||V_I|}{\Delta_A})^{\Delta_A})$ which is unacceptable in practice. Therefore, we model the graph poisoning policy as a Markov decision process.

Proposition 3.3 (Valid poisoning policy existence). *Suppose graph injection attacks are concerned. Given a benign graph $G = \{A, X\}$ with the node set V, a victim model f_ω and the budget $\Delta(G) = (\Delta_A, \Delta_X)$ as a cardinality constraint. Assume the injected node set is V_I, there exists at least one (G, f_ω)-valid poisoning policy with an upper bound of the complexity $O((\frac{e|V||V_I|}{\Delta_A})^{\Delta_A})$.*

Markov decision process. According to Definition 3.1, a valid policy can be regarded as a decision-making process from the initial state (*i.e.*, G) to the terminal state (*i.e.*, G^*) by sequentially inserting edges between the injected nodes and the partially poisoned graph. To find a more efficient method better than the brute-force search policy, we consider the sequential poisoning process as an episodic Markov decision process $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ are the set of states and actions respectively, P is the state transition function, r is the reward function and γ is a discount factor. Aligned with the threat model, at each time step t, we use the intermediate poisoned graph G'_t as the state. Since V_I is defined beforehand, the action can be generally regarded as the link prediction between nodes in V_I and G'_t. For the state transition function, as the state is fully observed, unexpected actions that violate the attack budget and unnoticeability requirement will be considered as being invalid and rejected. As described earlier, the central task of graph poisoning attacks is to find a policy π_θ that for each initial state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$ maximizes the expected return:

$$\max_{\theta} \mathbb{E}_{\pi_\theta}\Big[\sum_{t \geq 0} \gamma^t r_t \mid s, a\Big]. \tag{3}$$

For each time step t, we expect to maximize the generalization loss (*i.e.*, ℓ_{test}) to degrade the test accuracy. Since ℓ_{test} is not available to *Eve* either, we use the attack loss ℓ_{attack}, which is estimated by the surrogate model on the test data via self-training [12, 51, 58], to estimate the generalization loss. As different loss functions may lead to numerical differences, we directly use the accuracy relevant to the attack loss for the reward design. Therefore, the reward r at each time step is supposed to be a function *w.r.t.* the accuracy on unlabelled nodes, which is defined as follows:

$$r_t = \mathrm{Acc}_{attack}(f_{\omega^*}(G)) - \mathrm{Acc}_{attack}(f_{\omega^*_t}(G'_t)), \tag{4}$$

where $\mathrm{Acc}_{attack}(\cdot) \in [0, 1]$ returns the accuracy on the test set related to pseudo-labels and the attack loss. Unlike reward designs in previous works that may lead to sparse rewards [16] or inaccurate estimations between steps [49], we use the difference between Acc_{attack} on the (fixed) benign graph G and the partially poisoned graph G'_t at time step t, to represent the attack effectiveness.

Policy network. As is shown in Eq. (1), we use graph convolutional networks [30] for the representation $Z_{G'_t}$ of the partially poisoned graph G'_t. To accomplish the link prediction task, at the time step t the policy π_θ is supposed to select two nodes according to the representation $Z_{G'_t}$, which requires $O(|V||V_I| + |V_I|^2)$ by default. Previous methods [16, 49] mitigate this issue by hierarchically decomposing the single action into an action sequence, at the cost of lengthened episodes. On the complexity of the link prediction in node injections, we propose a naive policy χ to select the first node from V_I following a given distribution. To facilitate π_θ, the advantages of setting up χ are two-fold: a) π_θ only needs to select another node from the partially poisoned graph G'_t, which just requires $O(|V| + |V_I|)$; b) χ can follow some simple distributions, to minimize extra overheads and satisfy different requirements in certain scenarios, *e.g.*, the unnoticeability requirement.

To avoid adding all edges to a single node abnormally, we take χ as the uniform distribution over V_I by default, which imposes the average degree of the original graph on each injected node when $|V|\Delta_A = ||A||_0\Delta_X$ (*i.e.*, the budgets for nodes and edges are equivalent by ratio). We also take χ as the original node degree distribution to suffice to the unnoticeability requirement [76, 77]. Although χ can be viewed as a constraint on the power of graph poisoning policy, in the meanwhile χ greatly simplify the hypothesis space for π_θ, making π_θ learn the adversarial knowledge in a much more efficient manner. In Section 4.2 we empirically prove that χ not only brings us high flexibility to adapt to different scenarios, but also merely has a negligible impact on the final poisoning results. Formally, π_θ maps the graph embedding $Z_{G'_t}$ and the embedding *w.r.t.* the first node $u_t \sim \mathbb{P}_\chi(u)$ altogether to the second chosen node. We adopt Proximal Policy Optimization (PPO) [43] to instantiate π_θ by optimizing the following objective function:

$$\arg\max_\theta \ \mathbb{E}\left[\min\left(\frac{\pi_\theta}{\hat{\pi}_\theta}\mathcal{V}_t, \mathrm{clip}\left(\frac{\pi_\theta}{\hat{\pi}_\theta}, 1-\epsilon, 1+\epsilon\right)\mathcal{V}_t\right)\right], \tag{5}$$

where $\frac{\pi_\theta}{\hat{\pi}_\theta}$ is the probability ratio *w.r.t.* the old policy $\hat{\pi}_\theta$ and new policy π_θ which is clipped in $[1-\epsilon, 1+\epsilon]$, and $\mathcal{V}_t$ is the estimated advantage function at time step t. After the first and the second nodes are determined by χ and π_θ respectively, a new edge a_{uv} between the injected node u and node v from the poisoned graph is inserted, and the state is transited by the transition function P.

Lemma 3.4. *Given a graph distribution $\mathcal{G}$ and a victim model f_ω, a poisoning policy π_θ is $\mathcal{G}$-agnostic (Definition 3.2), if for any $G \sim \mathbb{P}_\mathcal{G}(G)$, π_θ is (G, f_ω)-valid.*

As is described in the threat model (Section 3.2), it is more practical to propose a $\mathcal{G}$-agnostic poisoning attack, since in real-world applications the training graph may change frequently, *e.g.*, friendship requests in the social network, especially in the transductive learning setting. Taking a closer look at Eq. (2), given f_ω, a poisoning policy can satisfy Definition 3.2 by iteratively checking if it is (G, f_ω)-valid for any $G \sim \mathbb{P}_\mathcal{G}(G)$ (Lemma 3.4). However, learning a $\mathcal{G}$-agnostic policy is even harder, since Lemma 3.4 requires the trained policy to have a stronger generalization ability over $\mathcal{G}$.

3.4 Curricula for Bi-level Optimization

In this part, we stand for the perspective of poisoning attackers, and formally introduce curricula for bi-level optimization in graph poisoning attacks, where the inner optimization attempts to minimize the training loss *w.r.t.* model parameters while the outer optimization tries to maximize the attack loss *w.r.t.* the poisoned training graph. The curricula in our proposed method cover two aspects: curricula in the inner optimization and curricula in the outer optimization. Within curricula, our method intends to learn the basic adversarial knowledge *w.r.t.* slightly perturbed graphs first, and then learn the advanced knowledge for a more destructive policy.

Inner optimization curricula. Considering the reward acquisition in Eq. (4), it is extremely time-consuming to go through the whole inner optimization from scratch at each time step of an episode, which makes the poisoning policy impractical for the training graph at scale. Since our adversarial purpose is to attack the training graph, *i.e.*, to find a (G, f_ω)-valid poisoning policy (Definition 3.1), evidently, for the inner optimization, the easiest task is optimizing on the clean graph, while the hardest task is finding the optimal poisoned graph and optimizing for it. Motivated by this, we design a burn-in stage for the inner optimization by pre-training on the clean graph at the beginning:

$$\tilde{\omega} = \arg\min_\omega \ell_{train}(f_\omega(G)), \tag{6}$$

where we let f_ω firstly be trained on the benign graph G to find the pre-trained parameters $\tilde{\omega}$ and keep them for further usage. These parameters $\tilde{\omega}$ make it possible to quickly transfer the learned clean knowledge into different adversarial knowledge during the inner optimization, with only a few GNN training epochs and a smaller learning rate re-scaled by the inner curriculum factor $\lambda \in (0, 1]$.

λ is set to 0.1 by default. Also, we consider different curricula in the inner optimization by setting smaller λ within the range of $(0, 1]$ for faster convergences. As was anticipated, later we empirically prove that these inner optimization curricula enable CuGPo to poison a wide range of training graphs in a more efficient manner.

Outer optimization curricula. As proved in Proposition 3.3 there is at least one (G, f_ω)-valid policy, so we can start by thinking about what tasks are easy tasks for the brute-force search policy. On the one hand, since the quality of curricula depends on the quality of poisoning sub-tasks, and further the quality of sub-tasks depends on the capacity of knowledge the poisoning policy can learn, it is natural to design curricula from the perspective of knowledge capacity, or attack budgets $\Delta(G) = (\Delta_A, \Delta_X)$. On the other hand, for the complexity $O((\frac{e|V||V_I|}{\Delta_A})^{\Delta_A})$ in Proposition 3.3, as usually $\Delta_A \ll |V||V_I|$ in order to suffice to unnoticeability, the budgets Δ_A dominate the complexity. Therefore, to generate a set of intermediate sub-tasks for adversarial knowledge transferring, we divide the final poisoning task under budgets $\Delta(G)$ into a set of sub-tasks C under different budgets. That is because the budgets serve as an important factor that determines the difficulty of using the brute-force search policy to solve the tasks. Back to the Markov decision process, at each step in an episode, a certain zero-entry a_{uv} in the

Algorithm 1 Curriculum Graph Poisoning (CuGPo)

```
Require: surrogate model $f_\omega$, number of training epochs $E$, inner
    curriculum factor $\lambda$, outer curriculum task set $C$
 1: Initialize the training graph $G$ and surrogate model $f_\omega$ with
    parameters $\omega$;
 2: for $i = 1$ to $E$ do
 3:     Compute the negative log-likelihood loss $\ell_{train}(f_\omega(G))$;
 4:     Update by descending the gradient: $\nabla_\omega \ell_{train}$ to get $\tilde{\omega}$;
 5: end for
 6: for $i = 1$ to $|C|$ do
 7:     Choose a curriculum task $C_i \in C$ in order;
 8:     Update candidate actions in $\pi_\theta$ according to $\Phi_i$;
 9:     for $t = 1$ to $|\Delta_i(G)|$ do
10:         Sample $G'_t \sim \mathbb{P}_{\pi_\theta}(G \mid f_\omega)$;
11:         Use $\tilde{\omega}$ to initialize surrogate model parameters $\omega$;
12:         for $j = 1$ to $\lfloor \lambda E \rfloor$ do
13:             Compute the loss $\ell_{train}(f_{\tilde{\omega}}(G'))$;
14:             Update by descending the gradient: $\nabla_{\tilde{\omega}} \ell_{train}$;
15:         end for
16:         Calculate $r_t$ via Eq. (4);
17:         Update $\theta$ using obtained experiences via Eq. (5);
18:     end for
19: end for
20: Return $\pi_{\theta^*}$ with optimized parameters $\theta^*$.
```

partially-poisoned adjacency matrix A' is set to 1 (*i.e.*, $a_{uv} \leftarrow 1$). Accordingly, the lower budgets represent shorter episodes, and we generate sub-tasks in the ascending order of the episode length. For the i-th sub-task $C_i \in C$ we modify Eq. (2) as:

$$\begin{aligned} \max_{G' \sim \mathbb{P}_{\pi_\theta}(G|f_\omega)} \quad & \ell_{test}(f_{\omega^*}(G')) \\ \text{s.t.} \quad & \omega^* = \arg\min_{\tilde{\omega}} \ell_{train}(f_{\tilde{\omega}}(G')) \\ & ||A' - A||_0 \le \Delta_{A,i}, \end{aligned} \tag{7}$$

where we have $0 < \Delta_i(G) \le \Delta_{i+1}(G), i = 1, \dots, |C| - 1$ and $\Delta_{|C|}(G) = \Delta(G)$. In the very beginning, when given the inner factor $\lambda \in (0, 1]$ CuGPo learns pre-trained parameters $\tilde{\omega}$ on the benign training graph G from scratch. Then starting from the easier task with lower budgets, CuGPo gradually learns to generate poisoned graphs from basic adversarial knowledge to advanced knowledge, using $\tilde{\omega}$ to initialize parameters in every inner optimization.

Note that since it is hard to precisely measure the difficulty of a specific poisoning task, we discuss the brute-force search policy to estimate the difficulty of the poisoning tasks, instead of directly using the brute-force policy to assess the learned policy. For other curricula in the outer optimization, we can consider a progressive action space, as $|V||V_I|$ *w.r.t.* the brute-force policy complexity $O((\frac{e|V||V_I|}{\Delta_A})^{\Delta_A})$ in Proposition 3.3. In this curriculum setting, the action space is divided into several parts. As steps go on, the action space is enlarged progressively. Assume $\Phi \in \mathbb{Z}$, there are $0 < \Phi_i \le \Phi_{i+1}, i = 1, \dots, |C| - 1$ and $\Phi_{|C|} = |V||V_I|$. The poisoning policy training process of CuGPo is formally described in Algorithm 1, and the proposed CuGPo is empirically evaluated in the following.

4 EXPERIMENTS

We investigate a variety of neural architectures including the graph convolutional networks (GCN) [30], GraphSAGE [25], graph attention networks (GAT) [52], graph isomorphism networks (GIN) [62] and GCNII [14]. For robust neural networks against graph poisoning attacks, we select two state-of-the-art architectures namely GCNLFR [10] and robust GCN (RGCN) [73]. For node classification datasets, we refer to previous graph poisoning studies [49, 78] and run experiments on the classical citation network datasets, including Cora [6, 40], Citeseer [22] and Pubmed [45]. We also select datasets with a larger scale, the co-purchasing network Amazon Photo and Amazon Computer [47] for node classification tasks. Since different curricula may lead to different experiment results, we refer to CuGPo with inner curriculum factor $\lambda = 0.1$ as CuGPo-L. However, in many cases, even $\lambda = 0.1$ is too large to perform poisoning attacks in due time. With a smaller inner curriculum factor λ, we also refer to CuGPo with progressive action spaces Φ and progressive budgets $\Delta(G)$ as CuGPo-SA and CuGPo-SB, respectively. By default, the number of sub-tasks in the outer optimization is set as $|C| = 2$ *w.r.t.* CuGPo-SA and CuGPo-SB.

Baselines. Likewise, empirical evaluations are performed between our proposed method and several graph poisoning methods, aligned with the threat model. a) Random: We consider a trivial attack based on a random policy, which inserts edges between injected nodes and the partially poisoned graph randomly. b) DICE [57]: As a heuristic attack strategy, it only removes and inserts edges between nodes from the same and different classes, respectively. We consider edge insertions between injected nodes and the partially poisoned graph according to the threat model. c) MetaGIA [15, 78]: It treats the input graph as a hyper-parameter and uses meta-learning to optimize the poisoned graph, as such meta parameter can fit on the final poisoned graph itself. For node injections, the edges of injected nodes are optimized. We consider the first-order approximated variant for efficiency. d) NIPA [49]: By inserting edges conditioned on injected nodes, NIPA is a graph poisoning attack method based on hierarchical reinforcement learning. As mentioned before, since there are some significant differences between poisoning attacks and evasion attacks, some evasion attacks [15, 75] are less appropriate for comparison: a) evasion attacks are performed at inference time, while the attack scope of our paper focuses on poisoning attacks at training time. One of the major factors that determine a successful poisoning attack is the unnoticeability requirement, *e.g.*, to keep the same node degree distribution [77, 78]. However, for evasion attacks [15, 75], the unnoticeability requirement in evasion attacks is not as important as in poisoning attacks [15], since queries at inference time are usually controlled by users. The difference in the unnoticeability requirement will impose different constraints on the attacks, making it less reasonable for comparison; b) evasion attacks are usually performed in the inductive learning setting, where the test instances will never be seen during training time [15], which is inconsistent with the transductive learning setting used in our paper as well as previous poisoning works [49, 78].

Table 1: Graph poisoning attacks against different datasets. We adopt different curricula for our proposed CuGPo. CuGPo-L and CuGPo-S indicate only curricula in inner optimizations applied. Each value indicates the test accuracy of graph convolutional networks for the node classification tasks. $^{\downarrow}$ indicates that the lower value represents better poisoning attack performance. † represents node injections only, without new edge-insertions. * indicates our proposed method with an oracle.

	Cora(↓)	**Citeseer(↓)**	**Pubmed(↓)**	**Photo(↓)**	**Computer(↓)**
Clean	84.70% ± 0.91%	76.06% ± 1.06%	88.00% ± 0.18%	91.77% ± 3.69%	86.22% ± 7.49%
Isolated Nodes†	84.92% ± 0.82%	77.10% ± 0.99%	88.14% ± 0.74%	91.59% ± 3.46%	86.55% ± 7.36%
Random	82.32% ± 0.65%	73.28% ± 1.04%	82.50% ± 0.51%	86.11% ± 1.09%	82.20% ± 0.37%
DICE	81.52% ± 1.07%	72.26% ± 0.26%	83.18% ± 0.14%	85.90% ± 1.44%	81.69% ± 0.64%
CuGPo-L	78.90% ± 0.60%	70.58% ± 0.42%	84.58% ± 1.48%	86.43% ± 1.18%	79.68% ± 1.75%
CuGPo-S	78.96% ± 0.43%	70.50% ± 0.66%	74.36% ± 0.66%	84.54% ± 1.01%	79.26% ± 1.16%
*CuGPo-S	79.94% ± 0.36%	71.38% ± 0.64%	74.78% ± 0.24%	87.12% ± 0.48%	79.46% ± 1.28%
CuGPo-SA	82.32% ± 0.42%	74.70% ± 0.33%	71.78% ± 1.15%	86.76% ± 0.69%	80.27% ± 1.11%
CuGPo-SB	77.48% ± 0.91%	69.44% ± 0.74%	74.28% ± 0.42%	85.37% ± 1.92%	78.64% ± 1.10%

4.1 Empirical Performance

By default, all experiments are repeated 5 times for statistical significance. The uncertainty in experiments represents the 95% confidence interval of the mean obtained via the student's t-test. According to the threat model, we report the test accuracy of node classification tasks corresponding to the best validation accuracy, which is known to both adversaries and victims. We use a two-layer GCN for graph representations when training, and keep the original settings for baseline methods. The same surrogate model is used to generate pseudo-labels for all methods. Also, in Table 1 CuGPo with an oracle represents our proposed method with access to all test node ground-truth labels. Note that this oracle only serves as a reference for investigating CuGPo in some extremely rare cases. By default all methods are under 5% budgets, *i.e.*, 5% injected nodes with 5% new edge-insertions to keep the original average degree of the training graph. During policy learning, we adopt early-stopping [1, 42] when the episodic reward no longer increases in a period.

Poisoning attacks. For victim model training, we run 200 epochs for Cora and Citeseer, 500 epochs for Pubmed and Amazon Photo, and 1000 epochs for Amazon Computer. The detailed dataset statistics and victim model set-ups are provided in the appendix. According to the threat model, adversaries are allowed to inject malicious nodes and insert new edges, without manipulating the existing edges of the original graph structure. Since in certain scenarios the features and labels are obtained via handcrafted rules and human supervision, following the common setting [29] we assume adversaries have no control over the features and labels of the injected nodes. For injected nodes, randomly generated labels are assigned to these nodes, and the features are based on the mean of existing features with noises sampled from the standard normal distribution. Table 1 shows that by injecting nodes alone without new edge establishments, the victim model does not suffer a significant accuracy drop on most datasets. We observe that non-learnable poisoning methods, *e.g.*, Random and DICE, are relatively fast to attack node classification tasks. However, for these methods, the test accuracy only suffers a minor drop and DICE is slightly more harmful than Random. For our proposed CuGPo, the learned policy is far beyond Random and DICE on most datasets, especially with further curricula in outer optimization. On citation networks, CuGPo with outer optimization curricula in budgets (*i.e.*, CuGPo-SB) is more harmful than CuGPo with inner optimization curricula alone. For larger datasets (*e.g.*, Amazon Photo), it is better to set smaller λ for faster convergence, since even in the case where $\lambda = 0.1$ (*i.e.*, CuGPo-L) the inner optimization is still time-consuming. Therefore, Random and DICE are indeed more efficient, yet they are only applicable to a very limited range of scenarios. Ordinarily, a surrogate model was used to estimate the generalization loss on the unlabelled nodes, since the adversary has no prior knowledge about the test set either. However, it is observed that there is no essential difference for CuGPo with or without an oracle, and the accuracy drops even slightly decrease due to the oracle, which is generally consistent with experimental results in the previous work [78].

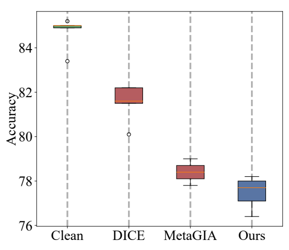
As MetaGIA needs to optimize the entire partially poisoned graph, it is quite memory-inefficient for larger graphs. We tried to run experiments of MetaGIA on graphs with more than 1×10^4 nodes (*e.g.*, Pubmed), yet failed to do so because it constantly triggers the *out-of-memory* error during the poisoning task. Thus we compare our proposed CuGPo with MetaGIA on Cora and Citeseer. Since the existing graph structure (stored in the database) is not allowed to modify, MetaGIA cannot always choose the optimal edges to establish if such edges already exist in the original graph. As a result, on some datasets (*e.g.*, Citeseer) MetaGIA is even worse than DICE, as is shown in Figure 1b. Without the curriculum setting, for NIPA the inner optimization is rather expensive. The estimated time for NIPA on a larger dataset (*e.g.*, Amazon Photo) is more than 30 days, thus we evaluate NIPA on Cora and Citeseer too. As is depicted in Figure 1c and Figure 1d, we evaluate every 2×10^4 steps for NIPA and every 1×10^5 steps for our proposed CuGPo. Even when our proposed CuGPo reaches the early-stopping step, NIPA is approximately much slower and has not finished the first 2×10^4 steps. Experiments against baselines reveal that with curricula our proposed CuGPo consistently outperforms MetaGIA on these datasets, and CuGPo is prominently more efficient than NIPA.

Generalization Performance. In Table 2 we evaluate our proposed CuGPo across different victim models, including vanilla GNNs and robust GNNs. Note that we assume CuGPo has no prior knowledge about the model architectures chosen by victims, this evaluation can measure the generalization performance of CuGPo over different victim models. Empirical results reveal that, with smaller λ for

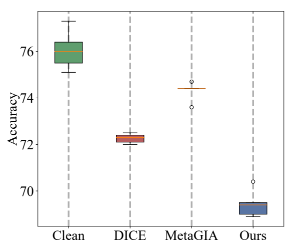
Table 2: Attack performance over different victim models. Values outside and inside the bracket represent the poisoned test accuracy by CuGPo and the clean accuracy, respectively. Unless stated otherwise, uncertainty error bars of clean test accuracy values are smaller than 5% by default. $\downarrow$ indicates that the lower value represents better poisoning attack performance.

	Cora($\downarrow$)	**Citeseer($\downarrow$)**	**Pubmed($\downarrow$)**	**Photo($\downarrow$)**	**Computer($\downarrow$)**
GCN	78.96% (84.70%)	70.50% (76.06%)	74.36% (88.00%)	84.54% (91.77%)	79.26% (86.22%)†
GraphSAGE	72.90% (85.06%)	65.04% (77.56%)	62.50% (89.80%)	75.29% (91.77%)	71.33% (83.28%)
GAT	71.76% (84.96%)	68.14% (79.46%)	60.58% (85.10%)	80.89% (88.48%)	67.68% (88.18%)
GIN	71.66% (74.62%)	61.76% (71.18%)	61.66% (85.62%)	88.33% (93.20%)	85.30% (88.97%)
GCNII	71.72% (85.82%)	63.44% (76.88%)	57.18% (88.60%)	76.32% (93.88%)	76.64% (90.71%)
RGCN	82.92% (85.62%)	71.96% (77.06%)	72.06% (85.80%)	27.79% (33.31%)†	37.35% (32.80%)†
GCNLFR	76.32% (86.02%)	67.64% (76.84%)	70.34% (87.52%)	82.96% (91.83%)	78.17% (86.51%)

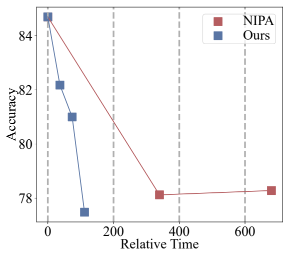
† represents values with error bars larger than 5%.



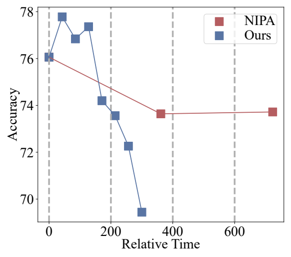
(a) MetaGIA on Cora



(b) MetaGIA on Citeseer



(c) NIPA on Cora



(d) NIPA on Citeseer

Figure 1: Classification test accuracy (%) under different poisoning methods. (a), (b) Comparing with MetaGIA; (c), (d) Comparing with NIPA. Ours denote CuGPo-SB.

efficiency, CuGPo can transfer the learned adversarial knowledge within GCN to various victim models. To our surprise, CuGPo is even more damaging on undefended GNNs other than the GCN as the surrogate model. For instance, compared with GCN, GCNII seems to be more vulnerable to our proposed CuGPo, especially on the Pubmed dataset. Among robust GNNs, RGCN is indeed more robust than vanilla ones. However, on Amazon Photo and Amazon Computer RGCN is very unstable, making its clean accuracy too low to serve as a decent node classifier. Empirical results indicate that GCNLFR is equivalent to or even more vulnerable than the vanilla GCN, and we attribute this vulnerability to the node injections caused by CuGPo that have undermined the assumption of robust eigenvalue intervals [10]. According to the results, our CuGPo has a strong generalization ability over various victim models.

Since in the transductive learning setting the training graph of interest is likely to update regularly, a graph-agnostic poisoning policy (Definition 3.2) is needed in practice. To create a group of graphs that share similar statistical characteristics, in this part, we forge a graph distribution by randomly rewiring the edges in the existing graph structure, and perform graph injection attacks based on the perturbed graph. As is depicted in Figure 3a and Figure 3b, we randomly rewire the training graph under certain perturbation rates to evaluate our proposed CuGPo. We perturb the training graph by 2%, 4%, 6%, 8% and 10% respectively. It is observed that when the perturbation rate $< 6\%$, our proposed CuGPo is still harmful in most cases, except for attacking Cora under the 2% perturbation rate. When the perturbation rate $\geq 6\%$, the perturbation itself can be taken as a poisoning method, since we restrict our attack budgets to 5%, and the alleged *clean* accuracy is very close to the poisoned accuracy in the unperturbed graph. Therefore, empirical results demonstrate that under certain budgets our proposed CuGPo is graph-agnostic *w.r.t.* the distribution by rewiring edges.

4.2 Poisoning Analysis

In this part, we analyze the poisoned graph under different settings of χ. Considering node injection attacks, where two nodes are selected to forge an edge between them, the gist of the hierarchical policy in previous methods [16, 49] is decomposing the single action into an action sequence (*i.e.*, choose one node first and then choose the second node based on the first node). Therefore, the hierarchical policy reduces the complexity from square level to linear level *w.r.t.* the number of nodes, on the cost of lengthened episodes. In this paper, we find out that the lengthened episodes can be avoided by sampling the first node from the injected node set V_I from a naive policy χ. In other words, there are actually two policies, namely the naive one χ and the learned one π_θ, during node injection attacks. As is depicted in Figure 3c and Figure 3d, on Cora in each time step we let χ sample the first node following a uniform distribution and the original node degree distribution, respectively. As suggested by previous work [77], since the node degree distribution generally resembles a power-law distribution alike shape in real networks, the victim can distinguish whether this graph is corrupted or not from the node degree distribution. When χ samples nodes following the degree distribution of the benign graph, the degree distribution of the resultant poisoned graph is similar to the original one, which indicates that our proposed CuGPo can effectively poison the benign graph under both the cardinality constraint and the node degree distribution constraint for unnoticeability. Also, owing to χ, the poisoning policy can learn the adversarial knowledge easily in the simplified hypothesis space. Figure 3d shows that before reaching the early-stopping step, two reward curves share a similar trend, which means the poisoning policy can quickly adapt to different χ.

Since we adopt the PPO algorithm based on policy gradients, it is vital to investigate the critical factors in hyper-parameter tuning. Figure 2 shows that for the first 3×10^5 steps, the reward curves vary according to the learning rates, which indicates the reward curves are somewhat sensitive to different hyper-parameters. Also, different hyper-parameters will have certain impacts on the resultant poisoning accuracy. We ascribe this observation to the adopted policy-based algorithm.

5 CONCLUSION AND DISCUSSIONS

In this paper, we propose a novel graph poisoning method against node classification tasks in the transductive learning setting. We explore the concept of *difficulty* in graph poisoning from the adversary's perspective, and propose CuGPo inspired by curriculum learning. We discuss the validity of a poisoning policy as well as the definition of the graph-agnostic poisoning policy, and empirically demonstrate the effectiveness of learning a poisoning policy in a progressive manner with extensive experiments.

Potentially, some malicious users could use CuGPo to attack a graph-based system, *e.g.*, social networks. Users with access to the training graph may register a few fake nodes to poison the targeted system. This paper intends to investigate the vulnerability of current GNNs, and hopefully promote awareness of the poisoning threat in the research community. For responsible disclosures, security researchers may utilize CuGPo to discover vulnerabilities within the graph-based system testing scope, and the security team of the target system could apply proactive security measures, *e.g.*, restricting user registration via Automated Turing Tests [53] and preventing users from getting the whole training graph.

Due to the intrinsic nature of the bi-level optimization in poisoning attacks, it is very hard to perform graph poisoning attacks on very large datasets. We have noticed that on larger datasets [27] all poisoning methods investigated in this paper will trigger serious errors, as well as some robust GNNs with higher complexity than vanilla ones. Also, as we adopt the reinforcement learning algorithm in our method, the choices of different hyper-parameters will have certain impacts on the resultant poisoned graph. We leave these issues along with the defense method for future directions.

ACKNOWLEDGMENTS

This work was supported by Beijing Natural Science Foundation (Z190001), Science and Technology Innovation 2030 - New Generation Artificial Intelligence (2020AAA0104401) and the National Key R&D Program of China (2018AAA0100702). Part of this work was supported by the National Innovation 2030 Major S&T Project of China (2020AAA0104203), and the Nature Science Foundation of China (62006007). We would like to sincerely thank the anonymous reviewers for valuable comments and suggestions.

REFERENCES

[1] Yingbin Bai, Erkun Yang, Bo Han, Yanhua Yang, Jiatong Li, Yinian Mao, Gang Niu, and Tongliang Liu. 2021. Understanding and Improving Early Stopping for Learning with Noisy Labels. In *NeurIPS*. 24392–24403.

[2] Aseem Baranwal, Kimon Fountoulakis, and Aukosh Jagannath. 2021. Graph Convolution for Semi-Supervised Classification: Improved Linear Separability and Out-of-Distribution Generalization. In *ICML (Proceedings of Machine Learning Research, Vol. 139)*. PMLR, 684–693.

[3] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *ICML (ACM International Conference Proceeding Series, Vol. 382)*. ACM, 41–48.

[4] Charles H. Bennett, Gilles Brassard, and Jean-Marc Robert. 1988. Privacy Amplification by Public Discussion. *SIAM J. Comput.* 17, 2 (1988), 210–229.

[5] Battista Biggio and Fabio Roli. 2018. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognit.* 84 (2018), 317–331.

[6] Aleksandar Bojchevski and Stephan Günnemann. 2018. Deep Gaussian Embedding of Graphs: Unsupervised Inductive Learning via Ranking. In *ICLR (Poster)*. OpenReview.net.

[7] Aleksandar Bojchevski and Stephan Günnemann. 2019. Adversarial Attacks on Node Embeddings via Graph Poisoning. In *ICML (Proceedings of Machine Learning Research, Vol. 97)*. PMLR, 695–704.

[8] Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. 2020. Scaling Graph Neural Networks with Approximate PageRank. In *KDD*. ACM, 2464–2473.

[9] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. *CoRR* abs/1606.01540 (2016).

[10] Heng Chang, Yu Rong, Tingyang Xu, Yatao Bian, Shiji Zhou, Xin Wang, Junzhou Huang, and Wenwu Zhu. 2021. Not All Low-Pass Filters are Robust in Graph Convolutional Networks. In *NeurIPS*. 25058–25071.

[11] Hong Chen, Yudong Chen, Xin Wang, Ruobing Xie, Rui Wang, Feng Xia, and Wenwu Zhu. 2021. Curriculum Disentangled Recommendation with Noisy Multi-feedback. In *NeurIPS*. 26924–26936.

[12] Jiefeng Chen, Frederick Liu, Besim Avci, Xi Wu, Yingyu Liang, and Somesh Jha. 2021. Detecting Errors and Estimating Accuracy on Unlabeled Data with Self-training Ensembles. In *NeurIPS*. 14980–14992.

[13] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *ICLR (Poster)*. OpenReview.net.

[14] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. 2020. Simple and Deep Graph Convolutional Networks. In *ICML (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 1725–1735.

[15] Yongqiang Chen, Han Yang, Yonggang Zhang, Kaili Ma, Tongliang Liu, Bo Han, and James Cheng. 2022. Understanding and Improving Graph Injection Attack by Promoting Unnoticeability. In *ICLR*. OpenReview.net.

[16] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. 2018. Adversarial Attack on Graph Structured Data. In *ICML (Proceedings of Machine Learning Research, Vol. 80)*. PMLR, 1123–1132.

[17] Minghong Fang, Neil Zhenqiang Gong, and Jia Liu. 2020. Influence Function based Data Poisoning Attacks to Top-N Recommender Systems. In *WWW*. ACM / IW3C2, 3019–3025.

[18] Minghong Fang, Minghao Sun, Qi Li, Neil Zhenqiang Gong, Jin Tian, and Jia Liu. 2021. Data Poisoning Attacks and Defenses to Crowdsourcing Systems. In *WWW*. ACM / IW3C2, 969–980.

[19] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *ICML (Proceedings of Machine Learning Research, Vol. 70)*. PMLR, 1126–1135.

[20] Jonas Geiping, Liam H. Fowl, W. Ronny Huang, Wojciech Czaja, Gavin Taylor, Michael Moeller, and Tom Goldstein. 2021. Witches' Brew: Industrial Scale Data Poisoning via Gradient Matching. In *ICLR*. OpenReview.net.

[21] Simon Geisler, Tobias Schmidt, Hakan Sirin, Daniel Zügner, Aleksandar Bojchevski, and Stephan Günnemann. 2021. Robustness of Graph Neural Networks at Scale. *CoRR* abs/2110.14038 (2021).

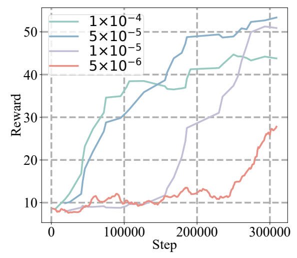
[22] C. Lee Giles, Kurt D. Bollacker, and Steve Lawrence. 1998. CiteSeer: An Automatic Citation Indexing System. In *ACM DL*. ACM, 89–98.

[23] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *ICLR (Poster)*.

[24] Guy Hacohen and Daphna Weinshall. 2019. On The Power of Curriculum Learning in Training Deep Networks. In *ICML (Proceedings of Machine Learning Research, Vol. 97)*. PMLR, 2535–2544.

[25] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*. 1024–1034.

[26] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. 2018. Rainbow: Combining Improvements in Deep Reinforcement Learning. In *AAAI*. AAAI Press, 3215–3222.



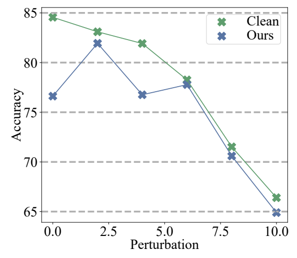
(a) Smoothed reward curves with different learning rates.

	Clean	LR=1×10^{-4}(↓)	LR=5×10^{-5}(↓)	LR=1×10^{-5}(↓)	LR=5×10^{-6}(↓)
GCN	84.70 ± 0.91	76.24 ± 0.55	77.90 ± 0.82	78.36 ± 1.04	79.22 ± 0.45
GraphSAGE	85.06 ± 0.71	73.06 ± 0.81	76.84 ± 0.52	74.46 ± 0.73	74.32 ± 0.62
GAT	84.96 ± 0.29	71.36 ± 1.39	76.32 ± 1.45	70.90 ± 1.46	71.36 ± 3.21
GIN	74.62 ± 3.60	69.96 ± 1.63	73.42 ± 2.01	75.20 ± 0.77	75.34 ± 0.72
GCNII	85.82 ± 0.72	71.48 ± 0.54	78.06 ± 0.55	75.04 ± 0.53	75.36 ± 0.95
RGCN	85.62 ± 0.20	80.42 ± 0.69	82.06 ± 0.55	82.74 ± 0.46	82.70 ± 1.20
GCNLFR	86.02 ± 0.41	75.42 ± 0.72	75.28 ± 0.49	75.20 ± 0.36	77.00 ± 0.93

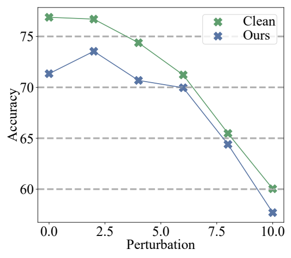
↓ denotes that the lower value represents better poisoning attack performance.

(b) Hyper-parameter tuning for learning rates. LR denotes the learning rate of PPO.

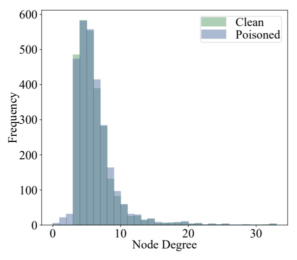
Figure 2: (a): Reward curves with different learning rates; (b) Poisoning performances with different learning rates. Each value indicates the test accuracy (%) for the node classification tasks.



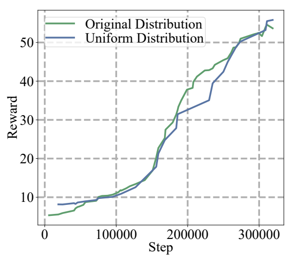
(a) Perturbed Cora



(b) Perturbed Citeseer



(c) Degree Distribution



(d) Reward Curve

Figure 3: (a), (b) Clean and poisoned test accuracy (%) under different perturbation rates (%); (c), (d) The node degree distributions and smoothed reward curves with different χ.

[27] 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*.

[28] W. Ronny Huang, Jonas Geiping, Liam Fowl, Gavin Taylor, and Tom Goldstein. 2020. MetaPoison: Practical General-purpose Clean-label Data Poisoning. In *NeurIPS*.

[29] Wei Jin, Yaxin Li, Han Xu, Yiqi Wang, and Jiliang Tang. 2020. Adversarial Attacks and Defenses on Graphs: A Review and Empirical Study. *CoRR* abs/2003.00653 (2020).

[30] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR (Poster)*. OpenReview.net.

[31] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper Insights Into Graph Convolutional Networks for Semi-Supervised Learning. In *AAAI*. AAAI Press, 3538–3545.

[32] Xixun Lin, Chuan Zhou, Hong Yang, Jia Wu, Haibo Wang, Yanan Cao, and Bin Wang. 2020. Exploratory Adversarial Attacks on Graph Neural Networks. In *ICDM*. IEEE, 1136–1141.

[33] Can Liu, Li Sun, Xiang Ao, Jinghua Feng, Qing He, and Hao Yang. 2021. Intention-aware Heterogeneous Graph Attention Networks for Fraud Transactions Detection. In *KDD*. ACM, 3280–3288.

[34] Xuanqing Liu, Si Si, Jerry Zhu, Yang Li, and Cho-Jui Hsieh. 2019. A Unified Framework for Data Poisoning Attack to Graph-based Semi-supervised Learning. In *NeurIPS*. 9777–9787.

[35] Zemin Liu, Qiheng Mao, Chenghao Liu, Yuan Fang, and Jianling Sun. 2022. On Size-Oriented Long-Tailed Graph Classification of Graph Neural Networks. In *WWW*. ACM, 1506–1516.

[36] Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. 2014. On the Computational Efficiency of Training Neural Networks. In *NIPS*. 855–863.

[37] Ben London and Lise Getoor. 2014. Collective Classification of Network Data. In *Data Classification: Algorithms and Applications*. CRC Press, 399–416.

[38] Jiaqi Ma, Shuangrui Ding, and Qiaozhu Mei. 2020. Towards More Practical Adversarial Attacks on Graph Neural Networks. In *NeurIPS*.

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

[40] Andrew McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. 2000. Automating the Construction of Internet Portals with Machine Learning. *Inf. Retr.* 3, 2 (2000), 127–163.

[41] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS*. 8024–8035.

[42] Garvesh Raskutti, Martin J. Wainwright, and Bin Yu. 2014. Early stopping and non-parametric regression: an optimal data-dependent stopping rule. *J. Mach. Learn. Res.* 15, 1 (2014), 335–366.

[43] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *CoRR* abs/1707.06347 (2017).

[44] Avi Schwarzschild, Micah Goldblum, Arjun Gupta, John P. Dickerson, and Tom Goldstein. 2021. Just How Toxic is Data Poisoning? A Unified Benchmark for Backdoor and Data Poisoning Attacks. In *ICML (Proceedings of Machine Learning Research, Vol. 139)*. PMLR, 9389–9398.

[45] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. 2008. Collective Classification in Network Data. *AI Mag.* 29, 3 (2008), 93–106.

[46] Ali Shafahi, W. Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. 2018. Poison Frogs! Targeted Clean-Label Poisoning Attacks on Neural Networks. In *NeurIPS*. 6106–6116.

[47] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of Graph Neural Network Evaluation. *CoRR* abs/1811.05868 (2018).

[48] Samarth Sinha, Animesh Garg, and Hugo Larochelle. 2020. Curriculum By Smoothing. In *NeurIPS*.

[49] Yiwei Sun, Suhang Wang, Xianfeng Tang, Tsung-Yu Hsieh, and Vasant G. Honavar. 2020. Adversarial Attacks on Graph Neural Networks via Node Injections: A Hierarchical Reinforcement Learning Approach. In *WWW*. ACM / IW3C2, 673–683.

[50] Fnu Suya, Saeed Mahloujifar, Anshuman Suri, David Evans, and Yuan Tian. 2021. Model-Targeted Poisoning Attacks with Provable Convergence. In *ICML (Proceedings of Machine Learning Research, Vol. 139)*. PMLR, 10000–10010.

[51] Kai Sheng Tai, Peter Bailis, and Gregory Valiant. 2021. Sinkhorn Label Allocation: Semi-Supervised Classification via Annealed Self-Training. In *ICML (Proceedings of Machine Learning Research, Vol. 139)*. PMLR, 10065–10075.

[52] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR (Poster)*. OpenReview.net.

[53] Luis von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. 2003. CAPTCHA: Using Hard AI Problems for Security. In *EUROCRYPT (Lecture Notes in Computer Science, Vol. 2656)*. Springer, 294–311.

[54] Xingchen Wan, Henry Kenlay, Binxin Ru, Arno Blaas, Michael A. Osborne, and Xiaowen Dong. 2021. Adversarial Attacks on Graph Classification via Bayesian Optimisation. *CoRR* abs/2111.02842 (2021).

[55] Jihong Wang, Minnan Luo, Fnu Suya, Jundong Li, Zijiang Yang, and Qinghua Zheng. 2020. Scalable attack on graph data by injecting vicious nodes. *Data Min. Knowl. Discov.* 34, 5 (2020), 1363–1389.

[56] Yunjuan Wang, Poorya Mianjy, and Raman Arora. 2021. Robust Learning for Data Poisoning Attacks. In *ICML (Proceedings of Machine Learning Research, Vol. 139)*. PMLR, 10859–10869.

[57] Marcin Waniek, Tomasz P Michalak, Michael J Wooldridge, and Talal Rahwan. 2018. Hiding individuals and communities in a social network. *Nature Human Behaviour* 2, 2 (2018), 139–147.

[58] Colin Wei, Kendrick Shen, Yining Chen, and Tengyu Ma. 2021. Theoretical Analysis of Self-Training with Deep Networks on Unlabeled Data. In *ICLR*. OpenReview.net.

[59] Qitian Wu, Hengrui Zhang, Junchi Yan, and David Wipf. 2022. Handling Distribution Shifts on Graphs: An Invariance Perspective. In *ICLR*. OpenReview.net.

[60] Zhaohan Xi, Ren Pang, Shouling Ji, and Ting Wang. 2021. Graph Backdoor. In *USENIX Security Symposium*. USENIX Association, 1523–1540.

[61] Chang Xu, Jun Wang, Yuqing Tang, Francisco Guzmán, Benjamin I. P. Rubinstein, and Trevor Cohn. 2021. A Targeted Attack on Black-Box Neural Machine Translation with Parallel Data Poisoning. In *WWW*. ACM / IW3C2, 3638–3650.

[62] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *ICLR*. OpenReview.net.

[63] Zhe Xu, Boxin Du, and Hanghang Tong. 2022. Graph Sanitation with Application to Node Classification. In *WWW*. ACM, 1136–1147.

[64] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. 2018. Hierarchical Graph Representation Learning with Differentiable Pooling. In *NeurIPS*. 4805–4815.

[65] Jaemin Yoo, Sooyeon Shim, and U Kang. 2022. Model-Agnostic Augmentation for Accurate Graph Classification. In *WWW*. ACM, 1281–1291.

[66] Bowen Zhang, Yidong Wang, Wenxin Hou, Hao Wu, Jindong Wang, Manabu Okumura, and Takahiro Shinozaki. 2021. FlexMatch: Boosting Semi-Supervised Learning with Curriculum Pseudo Labeling. In *NeurIPS*. 18408–18419.

[67] Hengtong Zhang, Yaliang Li, Bolin Ding, and Jing Gao. 2020. Practical Data Poisoning Attack against Next-Item Recommendation. In *WWW*. ACM / IW3C2, 2458–2464.

[68] Mengmei Zhang, Linmei Hu, Chuan Shi, and Xiao Wang. 2020. Adversarial Label-Flipping Attack and Defense for Graph Neural Networks. In *ICDM*. IEEE, 791–800.

[69] Sixiao Zhang, Hongxu Chen, Xiangguo Sun, Yicong Li, and Guandong Xu. 2022. Unsupervised Graph Poisoning Attack via Contrastive Loss Back-propagation. In *WWW*. ACM, 1322–1330.

[70] Zijie Zhang, Zeru Zhang, Yang Zhou, Yelong Shen, Ruoming Jin, and Dejing Dou. 2020. Adversarial Attacks on Deep Graph Matching. In *NeurIPS*.

[71] Tianxiang Zhao, Xiang Zhang, and Suhang Wang. 2022. Exploring Edge Disentanglement for Node Classification. In *WWW*. ACM, 1028–1036.

[72] Tianyi Zhou, Shengjie Wang, and Jeff A. Bilmes. 2020. Curriculum Learning by Dynamic Instance Hardness. In *NeurIPS*.

[73] Dingyuan Zhu, Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2019. Robust Graph Convolutional Networks Against Adversarial Attacks. In *KDD*. ACM, 1399–1407.

[74] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. 2020. Beyond Homophily in Graph Neural Networks: Current Limitations and Effective Designs. In *NeurIPS*.

[75] 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 *KDD*. ACM, 2461–2471.

[76] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. 2018. Adversarial Attacks on Neural Networks for Graph Data. In *KDD*. ACM, 2847–2856.

[77] Daniel Zügner, Oliver Borchert, Amir Akbarnejad, and Stephan Günnemann. 2020. Adversarial Attacks on Graph Neural Networks: Perturbations and their Patterns. *ACM Trans. Knowl. Discov. Data* 14, 5 (2020), 57:1–57:31.

[78] Daniel Zügner and Stephan Günnemann. 2019. Adversarial Attacks on Graph Neural Networks via Meta Learning. In *ICLR (Poster)*. OpenReview.net.

Table 3: Statistics of datasets for node classification. Directed graphs are transformed into undirected graphs for simplicity.

Dataset	Nodes	Edges	Training Nodes	Dimension	Classes
Cora	2708	13264	1208	1433	7
Citeseer	3327	12431	1827	3703	6
Pubmed	19717	108365	18217	500	3
Photo	7650	245812	3397	745	8
Computer	13752	505474	6107	767	10

A PROOFS

Proposition A.1 (Valid poisoning policy existence). *Suppose graph injection attacks are concerned. Given a benign graph $G = \{A, X\}$ with the node set V, a victim model f_ω and the budget $\Delta(G) = (\Delta_A, \Delta_X)$ as a cardinality constraint. Assume the injected node set is V_I, there exists at least one (G, f_ω)-valid poisoning policy with an upper bound of the complexity $O((\frac{e|V||V_I|}{\Delta_A})^{\Delta_A})$.*

Proof. According to the threat model, the poisoned node features $X' = \{x_u \mid u \in V_I \cup V\}$ are pre-defined. Since the benign graph $G = \{A, X\}$ is observed and deterministic and thus $\Delta(G)$ is deterministic, there is a finite solution set $\Omega = \{G' \mid ||A' - A||_0 \le \Delta_A\}$. Therefore, there always exists a poisoned graph $G^* \in \Omega$ that satisfies $\ell_{test}(f_{\omega^*}(G')) \le \ell_{test}(f_{\omega^*}(G^*))$ for any $G' \in \Omega$. To find G^*, considering graph poisoning attacks via the brute-force search policy, at most Δ_A edges can be inserted into the benign symmetric graph G and there are $\binom{|V||V_I|+0.5|V_I|(|V_I|+1)}{\Delta_A}$ choices. As it is still an attack if the number of inserted edges is less than Δ_A, we have

$$\sum_{i=0}^{\Delta_A} \binom{|V||V_I| + 0.5|V_I|(|V_I|+1)}{i} \tag{8}$$

$$\le \sum_{i=0}^{\Delta_A} \frac{(|V||V_I| + 0.5|V_I|(|V_I|+1))^i}{i!} \tag{9}$$

$$= \sum_{i=0}^{\Delta_A} \frac{{\Delta_A}^i(|V||V_I| + 0.5|V_I|(|V_I|+1))^i}{i!{\Delta_A}^i} \tag{10}$$

$$\le (\frac{|V||V_I| + 0.5|V_I|(|V_I|+1)}{\Delta_A})^{\Delta_A} \sum_{i=0}^{\Delta_A} \frac{{\Delta_A}^i}{i!} \tag{11}$$

$$\le (\frac{|V||V_I| + 0.5|V_I|(|V_I|+1)}{\Delta_A})^{\Delta_A} \sum_{i=0}^{\infty} \frac{{\Delta_A}^i}{i!} \tag{12}$$

where we have

$$(\frac{|V||V_I| + 0.5|V_I|(|V_I|+1)}{\Delta_A})^{\Delta_A} \sum_{i=0}^{\infty} \frac{{\Delta_A}^i}{i!}$$

$$= (\frac{e(|V||V_I| + 0.5|V_I|(|V_I|+1))}{\Delta_A})^{\Delta_A} \tag{13}$$

Since $\Delta_A \ll |V||V_I|$ and $|V_I| \ll |V|$ for the unnoticeability, there is

$$O((\frac{e(|V||V_I| + 0.5|V_I|(|V_I|+1))}{\Delta_A})^{\Delta_A}) = O((\frac{e|V||V_I|}{\Delta_A})^{\Delta_A}), \tag{14}$$

and thus the policy is bounded as $O((\frac{e|V||V_I|}{\Delta_A})^{\Delta_A})$.

□

Lemma A.2. *Given a graph distribution $\mathcal{G}$ and a victim model f_ω, a poisoning policy π_θ is $\mathcal{G}$-agnostic (Definition 3.2), if for any $G \sim \mathbb{P}_{\mathcal{G}}(G)$, π_θ is (G, f_ω)-valid.*

Proof. Given any $G \sim \mathbb{P}_{\mathcal{G}}(G)$ and $G^* \sim \mathbb{P}_{\pi_\theta}(G \mid f_\omega)$, if π_θ is (G, f_ω)-valid, according to Definition 3.1 there are $\ell_{test}(f_{\omega^*}(G')) \le \ell_{test}(f_{\omega^*}(G^*))$, $||A' - A||_0 \le \Delta_A$, and $||A^* - A||_0 \le \Delta_A$. Since the outer optimization in Eq. (2) aims to maximize $\ell_{test}(f_{\omega^*}(G'))$, there always exists G^* for any $G \sim \mathcal{G}$, and π_θ satisfies Definition 3.2.

□

B EXPERIMENTAL DETAILS

In this section, the experiment and implementation details of our proposed method are described. Most of our experiments are run on NVIDIA Tesla V100 with 32GB GPU memory. We use PyTorch [41] and OpenAI Gym [9] to implement our proposed method.

B.1 Model Architectures

Graph convolutional networks (GCN). The two-layer GCN [30] with a hidden size of 64 is used in the experiments. We set the Dropout probability as 0.5, and the activation function as ReLU. We use a two-layer GCN for graph representations when training, and use another GCN as a surrogate model for the inner optimization.

GraphSAGE. In the experiments, the two-layer GraphSAGE [25] with a hidden size of 64 is used. We set the Dropout probability as 0.5, and the activation function as ReLU.

Graph attention networks (GAT). For GAT [52] set-ups, the number of layers and the number of attention heads are 2 and 4, respectively. The hidden size is set as 64. We set the Dropout probability as 0.5, and the activation function as ELU and Leaky ReLU.

Graph isomorphism networks (GIN). The two-layer GIN [62] with a hidden size of 64 is used in the experiments. We set the Dropout probability as 0.5, and the activation function as ReLU.

GCNII. The hidden size of GCNII [14] is set as 64. We set the Dropout probability as 0.5, and the activation function as ReLU.

Robust GCN (RGCN). RGCN [73] is used for evaluating graph poisoning attacks against robust networks. In the experiments, the hidden size of RGCN is set as 64. We set the Dropout probability as 0.6. For activation functions, ELU is used for the mean and ReLU is used for the standard deviation *w.r.t.* the Gaussian distribution.

GCNLFR. GCNLFR [10] is used for evaluating graph poisoning attacks against robust nets. The two-layer GCNLFR with the hidden size of 128 is used in the experiments, with 0.5 Dropout probability. The probability for normal training and robust training is 0.5.

B.2 Dataset Detials

The statistics of datasets are listed in Table 3. We follow the splits proposed by [13] for citation network datasets (*i.e.*, Cora [6, 40], Citeseer [22] and Pubmed [45]), and all labels except those in the validation and test sets will be used for training. For Amazon Photo and Amazon Computer [47], the splits are randomly generated, with 3397 and 6107 nodes in the training set, 1414 and 2542 in the validation set, 2839 and 5103 nodes in the test set, respectively.