Query-Based and Unnoticeable Graph Injection Attack from Neighborhood Perspective

Chang Liu , Hai Huang * , Xingquan Zuo
Beijing University of Posts and Telecommunications
{changliu, hhuang, zuoxq}@bupt.edu.cn

Abstract

The robustness of Graph Neural Networks (GNNs) has become an increasingly important topic due to their expanding range of applications. Various attack methods have been proposed to explore the vulnerabilities of GNNs, ranging from Graph Modification Attacks (GMA) to the more practical and flexible Graph Injection Attacks (GIA). However, existing methods face two key challenges: (i) their reliance on surrogate models, which often leads to reduced attack effectiveness due to structural differences and prior biases, and (ii) existing GIA methods often sacrifice attack success rates in undefended settings to bypass certain defense models, thereby limiting their overall effectiveness. To overcome these limitations, we propose QUGIA, a Query-based and Unnoticeable Graph Injection Attack. QUGIA injects nodes by first selecting edges based on victim node connections and then generating node features using a Bayesian framework. This ensures that the injected nodes are similar to the original graph nodes, implicitly preserving homophily and making the attack more unnoticeable. Unlike previous methods, QUGIA does not rely on surrogate models, thereby avoiding performance degradation and achieving better generalization. Extensive experiments on six real-world datasets with diverse characteristics demonstrate that QUGIA achieves unnoticeable attacks and outperforms state-of-the-art attackers. The code can be found at https://github.com/1234238/QUGIA

1 Introduction

Graph Neural Networks (GNNs), as a representative approach for processing graph-structured data, have shown promising performance in tasks involving relational information [Jiang *et al.*, 2021; Xing *et al.*, 2024]. GNNs recursively learn feature and topological information through a message-passing paradigm. Nevertheless, this paradigm also leads to their vulnerability [Dai *et al.*, 2018; Zügner *et al.*, 2018], and many attack methods have been proposed to explore their robustness. Previous studies mainly focus on Graph Modification Attacks (GMA) [Zügner and Günnemann, 2019; Zügner *et al.*, 2018], which reduce the performance of GNNs by modifying node features and edge connections in the original graph. However, GMA is difficult to implement in practice because attackers often lack the privileges to alter existing data in a graph. To make graph attacks more practical, researchers have proposed Graph Injection Attacks (GIA) [Ju *et al.*, 2023; Chen *et al.*, 2022; Tao *et al.*, 2021; Zou *et al.*, 2021; Wang *et al.*, 2020; Sun *et al.*, 2020], which reduce the performance of GNNs by injecting malicious nodes.

The settings of GIA can be categorized into white-box and black-box approaches. White-box methods require access to the parameters and architecture of the target model, which is impractical in real-world scenarios. Therefore, we focus on GIA in the black-box setting. Although numerous effective GIA studies have emerged [Chen *et al.*, 2022; Zou *et al.*, 2021; Wang *et al.*, 2020; Sun *et al.*, 2020], this type of attack often disrupts the homophily of the original graph [Li *et al.*, 2023; Chen *et al.*, 2022]. Homophily indicates that neighboring nodes tend to have similar node features or labels, which is a key characteristic of homophilous graphs. Thus, GIA can be effectively defended against by homophily defenders using edge pruning [Zhang and Zitnik, 2020]. To address this problem, the Harmonious Adversarial Objective (HAO) [Chen *et al.*, 2022] enforces GIA to preserve the homophily of the original graph. Nonetheless, according to the no-free-lunch principle, we observe that HAO sacrifices attack performance against undefended models to enhance performance against homophily defenders. Moreover, existing GIA methods mainly rely on surrogate models to generate perturbed graphs. However, different architectures and prior biases pose significant challenges to the generalization ability of the attack method when applied to a new target model. Although recent works [Ju *et al.*, 2023; Wang *et al.*, 2021] attempt to mitigate the dependency on surrogate models by generating perturbed graphs without gradients and using query-based attacks, these methods remain susceptible to homophily-based defenses. These critical flaws raise a significant concern: *How can we design a surrogate-free attack method that minimizes damage to attack performance against undefended models while remaining undetectable by homophily defenders?*

To address the aforementioned challenges, we propose a

*Corresponding author

Query-based and Unnoticeable Graph Injection Attack (QUGIA), which is entirely independent of surrogate models. Following existing GIA methods, our approach consists of two main components: *feature generation* and *edge generation*. The feature generation component generates the features of the injected nodes, while the edge generation component determines the edges that connect the injected nodes to the original graph nodes. For the feature generation component, we adopt a Bayesian framework to generate the features of the injected nodes, allowing the model to learn from historical Bayesian inference information. We initialize the features of the injected nodes based on those of the victim nodes. Subsequently, we optimize the features of the injected nodes within a fixed perturbation range, K, which implicitly preserves the homophily of the original graph. For the edge generation component, previous work generally generates edges from the perspective of node degree, as nodes with lower degrees are less robust [Li *et al.*, 2023; Zou *et al.*, 2021]. Considering the message-passing paradigm and the homogeneity of the graph, unlike conventional methods, we generate edges from the perspective of the neighbors of the victim nodes. QUGIA avoids the notorious bi-level optimization problem, which involves alternating optimization between edge generation and node generation. Moreover, it implicitly preserves the homogeneity of the graph, making the attack less noticeable. QUGIA provides a novel perspective for the design of GIA attacks. Our contributions are summarized as follows:

- We investigate query-based GIA that are unnoticeable under defenses while effective against undefended models, which were rarely discussed in previous studies.
- We propose a query-based attack framework that is independent of specific model assumptions. Our approach introduces a Bayesian optimization-based method for updating injected node features and an injection strategy based on the victim node's neighbors.
- Extensive experiments on six real-world datasets and six GNN models demonstrate that our attack method outperforms state-of-the-art attackers.

2 Preliminary

2.1 Graph Neural Networks

Consider a graph $G = (V, E)$, where $V = \{v_1, \ldots, v_n\}$ represents the set of nodes, and $E = \{e_1, \ldots, e_m\}$ represents the set of edges. Let X denote the node feature matrix, with $X \in \mathbb{R}^{n \times d}$, where n is the number of nodes and d is the dimensionality of the node features. The feature vector of a node u is denoted by X_u. The adjacency matrix is represented as $A \in \{0, 1\}^{n \times n}$, where $A_{ij} = 1$ if there exists an edge between nodes i and j, and $A_{ij} = 0$ otherwise. The set of neighbors of a node v is denoted by $\mathcal{N}(v)$. In this paper, we focus on the semi-supervised node classification task. The nodes used during training are denoted as V_{train}. For each node u in V_{train}, there is a corresponding label $y_u \in Y_L$, where $Y_L \subseteq Y$ and $Y = \{1, 2, \ldots, C\}$. Y_L represents the set of labeled nodes. During the testing phase, the trained GNN model predicts the labels of the nodes in $V_{\text{test}} = V \setminus V_{\text{train}}$ based on the subgraph G_{test}.

2.2 Graph Adversarial Attacks

During the training phase, the parameters of the GNN model are learned using G_{train}. The objective of the adversarial attack is to construct a perturbed graph G' that reduces the classification accuracy of the trained GNN model on V_{test} during the testing phase. The general formulation of graph adversarial attacks can be expressed as follows:

$$\min \mathcal{L}_{\text{atk}}(f_\theta(G')), \text{ s.t. } \|G' - G\| \leq \Delta, \tag{1}$$

where θ represents the trained parameters of the GNN model, $\mathcal{L}_{\text{atk}}$ denotes the loss function for the attack (e.g., $-\mathcal{L}_{\text{train}}$), and Δ constrains the perturbation strength allowed for the attacker.

For GIA, the perturbed adjacency matrix and feature matrix can be expressed as follows:

$$A' = \begin{bmatrix} A & A_I \\ A_I^T & O_I \end{bmatrix}, \quad V' = \begin{bmatrix} V \\ V_I \end{bmatrix}, \tag{2}$$

where A_I denotes the adjacency matrix between the original nodes and the injected nodes, O_I represents the adjacency matrix among the injected nodes, and V_I corresponds to the injected nodes. The constraints for GIA attacks can be summarized as:

$$\begin{aligned} &|V' - V|_0 \leq \Delta, \quad 1 \leq d_u \leq b, \\ &\min(X) \leq X_{u,i} \leq \max(X), \quad \forall i \in \{1, 2, \ldots, d\} \end{aligned} \tag{3}$$

where Δ represents the maximum number of injected nodes, d_u refers to the degree of the injected node u, and $X_{u,i}$ denotes the feature value of the injected node u in the i-th dimension. The terms $\min(X)$ and $\max(X)$ represent the minimum and maximum values of all node features in the original graph after normalization. It is worth noting that discrete features are not normalized in this process.

Following previous works [Chen *et al.*, 2022; Zheng *et al.*, 2021], this paper primarily focuses on an evasion attack scenario where the attacker is not allowed to modify the model's parameters. The attack operates in an inductive learning setting, meaning that test nodes remain entirely unseen during training. Furthermore, the attack is characterized as non-targeted and black-box. Specifically, the attacker aims to reduce the model's accuracy by causing as many misclassifications as possible across the entire test set within a given attack budget, without targeting specific nodes or classes, and without any knowledge of the model's architecture or parameters.

3 Proposed Method

In this chapter, we provide a detailed introduction to our proposed attack method, QUGIA. Figure 1 illustrates the overall framework of QUGIA, which consists of two core components: node generation and edge generation. Node generation optimizes the features of injected nodes using Bayesian optimization, while edge generation determines the connections of injected nodes based on the neighbors of the victim node.

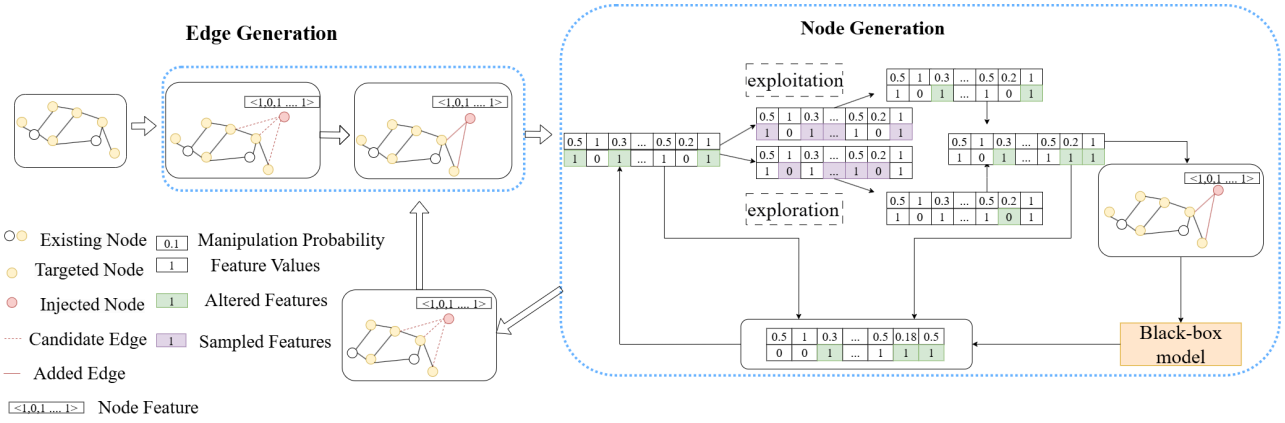

Figure 1: QUGIA consists of two components: edge generation and node generation. Edge generation first connects injected nodes to the original graph nodes by considering the neighborhood of the victim nodes, followed by node generation, which optimizes the features of the injected nodes using a novel Bayesian framework.

In each injection step, QUGIA first performs edge generation to establish the connections for the injected nodes, followed by the optimization of their features. As feature optimization is the primary aspect of the injection process, we will first discuss node generation, followed by edge generation.

3.1 Edge Generation

Generating malicious edges is a discrete problem with an extremely large search space due to the numerous possible connections in the graph. Alternating between feature optimization and edge selection leads to the notorious bi-level optimization problem, which is challenging to address within a limited number of queries. Existing works mainly tend to attack low-degree victim nodes because such nodes exhibit lower robustness compared to high-degree nodes [Li *et al.*, 2023; Zou *et al.*, 2021]. Although attacks from the perspective of node degree have been extensively studied, this approach suggests that the topological structure of the graph can be leveraged to simplify the design of attack methods. Unlike existing approaches, our work designs edge generation by considering the neighbors of the victim node, offering a novel perspective for GIA.

Due to the message-passing paradigm in graphs, changes in a node can affect its neighboring nodes. When a new node V_u' is injected to attack victim node V_u, the feature representation of V_u is disrupted, which in turn impacts its neighboring nodes. Therefore, we designed malicious edge generation from the perspective of neighbors, expanding the attack area to $V_u \cup \mathcal{N}(u)$. We only focus on first-order neighbors because the influence between graph nodes diminishes as the distance between them increases. Let p_u to denote the first-order neighbors of node in V_{test}, where $p_u = \mathcal{N}_1(u) \cap V_{test}$. The homophily of the graph implies that neighboring nodes (particularly first-order neighbors) are likely to exhibit similar behaviors. If we can implicitly ensure the similarity between V_u' and V_u, then the newly injected node and $\mathcal{N}(u)$ are likely to display a higher degree of homophily. This edge generation method not only simplifies the attack method but also implicitly preserves the homophily of the original graph.

We perform attacks in descending order based on $|p_u|$, where $|p_u|$ represents the number of nodes in the set p_u. If attacks are performed in ascending order based on $|p_u|$, when $|p_u| = 0$, our attack method degenerates into a single-node attack strategy. When attacking node u, $\forall u \in V_{test}$, we connect the malicious new node X_u' to k randomly selected nodes from p_u, ensuring that X_u' is always connected to X_u.

We utilize the Carlini-Wagner (CW) loss function [Xu *et al.*, 2019] for our loss component, which is expressed as follows:

$$\ell = \max(-\gamma,\ (f_\theta(G')_{y_u} - \max_{c \neq y_u} f_\theta(G')_c)) \tag{4}$$

Here, γ represents the confidence score of $\mathcal{L}_{\text{atk}}$. A higher γ implies a stronger attack effect but results in a greater distance from the decision boundary. To ensure the attack remains as close to the decision boundary as possible, we set $\gamma = 0.05$. Combining Eq. (1) and Eq. (4), the complete attack loss function is:

$$\ell = \sum_{u \in p_u} (\max(-0.05, (f_\theta(x_u')_{y_u} - \max_{c \neq y_u} f_\theta(x_u')_c))) \tag{5}$$

To maximize the utilization of the injection node edge budget, we record triplet data $(src, dst, score)$, where $src \in V_I$, $dst \in V$, and $score \in [0, 1]$. There is no existing connection between src and dst. The $score$ value, computed using Equation 4, represents the distance of the dst node from the decision boundary, with smaller values indicating a higher likelihood of a successful attack. After completing the attack process, if any injection node edge budget remains, the recorded triplets are sorted in ascending order based on their $score$ values, and connections are established by prioritizing the pairs with the lowest scores.

3.2 Node Generation

Since reliance on surrogate models is eliminated, generating the features of injected nodes cannot depend on neural networks and instead becomes a combinatorial optimization problem. The high dimensionality of graph data features significantly expands the search space, particularly for

continuous features, where the search space spans the range $[\min(X), \max(X)]$. To address this issue, we first reduce the dimensionality of the search space and then perform the search within the simplified space. This process presents two primary challenges: (i) determining the magnitude of feature updates and (ii) selecting the dimensions for feature updates.

The magnitude of feature updates refers to the degree of changes applied to the features during each update. In surrogate-based attacks, attackers can utilize gradients to assign varying update sizes at each step. However, in heuristic-based search, gradients are unavailable, and exhaustive search becomes impractical. To address this issue, we define feature updates as the distance between feature values and their boundary values. By doing so, continuous feature values are replaced with discrete, finite values, allowing perturbations within a significantly reduced search space. Specifically, the features of the injected node are defined as $X'_u = s\tilde{X}_u + (1-s)X_u$, where $\tilde{X}_u$ is defined in Eq. (6), and s indicates whether a specific position in the feature vector is flipped, with $s \in \{0,1\}^{1\times d}$.

$$\tilde{X}_u = \begin{cases} \min(X) & \text{if } X_u > \frac{\min(X)+\max(X)}{2} \\ \max(X) & \text{otherwise} \end{cases} \tag{6}$$

After determining the magnitude of feature updates, we focus on how to select the feature combinations for updates. Specifically, we aim to further reduce the search dimensionality of s. Following the inspiration from BBA [Lee *et al.*, 2022], which first identifies a successful adversarial sample and then searches for unnoticeable perturbations around it, we propose a novel approach from a different perspective. Instead of searching for perturbations around a successful adversarial sample, our goal is to initially identify the most unnoticeable injected node feature $X'_{\text{u\_init}} = X_u$, followed by searching within its neighboring feature space to find X'_u that ensures both a successful attack and unnoticeability. The subsequent challenge is how to define the neighboring feature space. Intuitively, the fewer modifications are made to $X'_{\text{u\_init}}$, the closer it remains to the original feature space, which aligns with the principles of sparse attacks [Vo *et al.*, 2024; Croce and Hein, 2019; Narodytska and Kasiviswanathan, 2017]. Based on this intuition, we define a relaxed search budget K, where $K \in \mathbb{Z}$, representing positive integers $\{1, 2, 3, \dots\}$. For each injected node feature X'_u, we modify only a fixed number of feature dimensions, thereby further reducing the search space. In search problems, the trade-off between exploration and exploitation is inevitable. To manage this trade-off, we employ a power decay strategy, as described in Eq. (7), where λ_t represents the exploration rate at iteration t, B denotes the exponential decay speed, and A controls the extent of the initial exploration. Here, A and B are real hyperparameters that can be tuned to balance the trade-off between exploration and exploitation effectively. Larger values of A promote broader initial exploration, while smaller values of B result in faster decay, leading to greater exploitation in later iterations.

$$\lambda_t = A \cdot B^t \tag{7}$$

It is evident that different features contribute variably to the success of the attack, making it reasonable to treat these features differently. To address this, we employ a Bayesian evolutionary algorithm to learn the influence of features from historical data and approximate their impact using a probabilistic framework. Specifically, we utilize a categorical distribution to simulate the influence of different feature dimensions, as selecting various feature combinations corresponds to multiple draws from a set of possible categories. We use the Dirichlet distribution as the prior for the categorical distribution, as it effectively models the probability distribution over each category or combination of categories. The categorical distribution is parameterized by θ, with its prior defined as: $P(\theta;\alpha) := \text{Dir}(\alpha)$, where $\alpha = [\alpha_1, ..., \alpha_d]$ represents the concentration parameters. A uniform distribution is employed with $\alpha_i = 1$. The Bayesian framework is structured into three stages: initialization, sampling, and updating.

Initialization of X'_u. We propose a sequential attack method. When attacking a victim node X_u, we initialize the features of the injected node X'_u using the features of X_u, i.e., $X'_u = X_u$. We randomly select K positions in s and set them to 1.

Sampling of s_i. We perform sampling based on θ and $s^{(t-1)}$, where $s_i^{(t-1)}$ denotes the value of the i-th dimension of the perturbation vector s at the $(t-1)$-th iteration. Eq. (8) focuses on exploring unknown perturbation combinations, while Eq. (9) focuses on exploiting known perturbation combinations.

$$s_1^{(t)}, \dots, s_{\lceil K\cdot\lambda_t \rceil}^{(t)} \sim \text{Categorical}(\mathbf{s} \mid \theta^{(t)}, s^{(t-1)} = 0) \tag{8}$$

$$s_{\lceil K\cdot\lambda_t+1 \rceil}^{(t)}, \dots, s_K^{(t)} \sim \text{Categorical}(\mathbf{s} \mid \theta^{(t)}, s^{(t-1)} = 1) \tag{9}$$

$$s^{(t)} = \left[\bigvee_{i=1}^{K\cdot\lambda_t} s_i^{(t)}\right] \vee \left[\bigvee_{j=K\cdot\lambda_t+1}^{K} s_j^{(t)}\right] \tag{10}$$

The exact solution for the underlying parameter distribution θ is typically unattainable. However, since the Dirichlet distribution is a conjugate prior, we can approximate θ using the expectation of the Dirichlet posterior distribution, which is learned and updated through Bayesian inference over time.

$$\alpha_i^{\text{posterior}} = \alpha_i^{\text{prior}} + s_i^{(t)} \tag{11}$$

$$\theta^{(t)} = \mathbb{E}_{\theta\sim P(\theta|\alpha,u^{(t-1)},\ell^{(t-1)})}[\theta] \tag{12}$$

The posterior concentration parameter $\alpha_i^{\text{posterior}}$ is calculated by adding the observation value $s_i^{(t)}$ from the t-th iteration to the prior concentration parameter α_i^{prior}. Specifically, $s_i^{(t)}$ is defined as: $s_i^{(t)} = \left(\frac{q_i^{(t)}+0.001}{v_i^{(t)}+0.001}\right)$, where $q_i^{(t)}$ represents the accumulated importance of the i-th dimension feature, as described in Eq. (13), and $v_i^{(t)}$ represents the accumulated access count for the feature at position i, as described

in Eq. (14). The loss function at the t-th iteration of feature optimization is denoted as: $\ell^{(t)} = \mathcal{L}_{\text{atk}}(f_\theta(G'))^{(t)}$.

$$q_i^{(t)} = \begin{cases} q_i^{(t-1)} + 1 & \text{if } \ell^t \geq \ell^{(t-1)} \\ & \wedge s_i^{(t)} = 1 \wedge s_i^{(t-1)} = 0 \\ q_i^{(t-1)} & \text{otherwise} \end{cases} \tag{13}$$

$$v_i^{(t)} = \begin{cases} v_i^{(t-1)} + 1 & \text{if } s_i^{(t)} = 1 \vee s_i^{(t-1)} = 1 \\ v_i^{(t-1)} & \text{otherwise} \end{cases} \tag{14}$$

Updating of $s^{(t)}$. We generate a sample $s^{(t)}$ during the t-th feature update, where each $s^{(t)}$ represents a candidate solution. If the new perturbed feature outperforms the perturbed feature from the $(t-1)$-th iteration, we retain the current sample. Otherwise, we retain the sample $s^{(t-1)}$ from the previous iteration. The corresponding formula is as follows:

$$s^{(t)} = \begin{cases} s^{(t)} & \text{if } \ell^{(t)} < \ell^{(t-1)} \\ s^{(t-1)} & \text{otherwise} \end{cases} \tag{15}$$

3.3 Comprehensive Execution Process of QUGIA

In this subsection, we provide a detailed explanation of the execution process of QUGIA. The complete execution procedure is presented in Algorithm 1. In each injection step, QUGIA first determines the connectivity of the injected node through edge generation (lines 2–3) and then optimizes its features (lines 4–17). After injecting a node, it updates the structural information (line 18) and repeats the process until the node injection budget is exhausted. If any edge injection budget remains, it is allocated through edge generation (line 20).

Algorithm 1 QUGIA

Input: Graph $G = (A, X)$, target node set V_t, injection node attack budget Δ_n, injection node edge budget Δ_e, maximum iterations T, Dirichlet parameters, structural selection score $|p|$, model-predicted label function $f()$, node labels Y.
Output: Final adversarial graph G'

1: **for** each $i \in [0, \Delta_n)$ **do**
2: Select victim node u based on sorted scores $|p|$
3: Initialize A'_u by connecting injected nodes to node u and its neighbors
4: **while** $t < T$ and $\exists j \in (p_u \cup \{u\}), f(j) \neq Y_j$ **do**
5: Initialize feature matrix $X_u'^{(0)}$
6: Sample perturbation vector $s_i^{(t)}$ according to Equations (8–10)
7: $X_u'^{(t)} \leftarrow s^{(t)}\tilde{X}_u + (1 - s^{(t)})X_u$
8: Calculate loss ℓ^t for $X_u'^{(t)}$ using Equation (5)
9: **if** $\ell^t < \ell^{t-1}$ **then**
10: $X_u'^{(t)} \leftarrow X_u'^{(t)}$
11: Update A', X' with A'_u, $X_u'^{(t)}$ using (2)
12: **else**
13: $X_u'^{(t)} \leftarrow X_u'^{(t-1)}$
14: **end if**
15: Update λ_t using Equation (7)
16: Update Dirichlet parameters according to Equations (11–14)
17: **end while**
18: If $f(u) \neq Y_u$, then for all $m \in \mathcal{N}_1(u) \cap V_t$, the score is updated as $|p_m| = |p_m| - 1$.
19: **end for**
20: Allocate remaining edge budget Δ_e
21: **return** $G' = (A', X')$

4 Experiments

4.1 Datasets

We evaluate our approach on six datasets, covering both discrete and continuous features. The discrete datasets include Cora and Citeseer [Yang *et al.*, 2016], while the continuous datasets comprise Pubmed [Sen *et al.*, 2008], the GRB-redefined versions of Cora and Citeseer [Zheng *et al.*, 2021], and the arXiv dataset from OGB [Hu *et al.*, 2020]. We adopt a data-splitting strategy similar to those used in prior studies [Chen *et al.*, 2022; Zheng *et al.*, 2021]. Detailed descriptions of the datasets are provided in the *Supplementary Material*.

4.2 Baseline

The study of node insertion attacks in black-box settings remains relatively unexplored. To provide a comprehensive evaluation, we selected several representative methods as baselines. These include TDGIA [Zou *et al.*, 2021], the state-of-the-art method for node insertion attacks; ATDGIA [Chen *et al.*, 2022], a variant of TDGIA; and AGIA [Chen *et al.*, 2022], which adopts a Gradient-Driven Injection strategy. Additionally, we incorporated HAO [Chen *et al.*, 2022] into TDGIA, ATDGIA, and AGIA to thoroughly assess the unnoticeability of the attacks. Since these methods require gradients from the victim model, we set the surrogate model to have the same architecture as the victim model, with a fixed random seed of 666. Furthermore, we included G2A2C [Ju *et al.*, 2023], a recent query-based attack that leverages reinforcement learning. To ensure a fair comparison, G2A2C was configured to operate under the same maximum number of single-node queries as our proposed method.

4.3 Evaluation Protocol

To ensure a comprehensive comparison, we selected three widely used GNN models without defense mechanisms and three GNN models with defense mechanisms. For models without defenses, we included GCN [Kipf and Welling, 2017], GAT [Veličković *et al.*, 2018], and APPNP [Gasteiger *et al.*, 2019]. For models with defenses, following prior work [Chen *et al.*, 2022], we selected Guard [Zhang and Zitnik, 2020], EGuard, and RGAT [Chen *et al.*, 2022].

The number of inserted nodes is defined as $|V_I| = a|V|$, where a represents the percentage of the total number of nodes in the clean graph. To evaluate the performance of various attack methods under different attack constraints, we set a to 1%, 3%, and 5%. The maximum degree of the inserted nodes is constrained by the average degree of the corresponding graph dataset.

Datasets	a	TDGIA	TDGIA+HAO	ATDGIA	ATDGIA + HAO	AGIA	AIGA+HAO	G2A2C	QUAGIA
Cora	0.01	$0.988_{\pm 0.001}$	$0.996_{\pm 0.001}$	$0.973_{\pm 0.002}$	$0.985_{\pm 0.001}$	$0.974_{\pm 0.003}$	$0.985_{\pm 0.002}$	$0.988_{\pm 0.002}$	$\mathbf{0.956}_{\pm 0.002}$
	0.03	$0.946_{\pm 0.004}$	$0.971_{\pm 0.003}$	$0.924_{\pm 0.003}$	$0.961_{\pm 0.003}$	$0.947_{\pm 0.003}$	$0.952_{\pm 0.007}$	$0.965_{\pm 0.005}$	$\mathbf{0.881}_{\pm 0.004}$
	0.05	$0.899_{\pm 0.009}$	$0.929_{\pm 0.005}$	$0.898_{\pm 0.005}$	$0.924_{\pm 0.006}$	$0.920_{\pm 0.006}$	$0.923_{\pm 0.004}$	$0.953_{\pm 0.004}$	$\mathbf{0.815}_{\pm 0.005}$
Citeseer	0.01	$0.991_{\pm 0.001}$	$0.998_{\pm 0.000}$	$0.981_{\pm 0.002}$	$0.991_{\pm 0.001}$	$0.982_{\pm 0.001}$	$0.992_{\pm 0.001}$	$0.993_{\pm 0.001}$	$\mathbf{0.967}_{\pm 0.002}$
	0.03	$0.973_{\pm 0.002}$	$0.993_{\pm 0.002}$	$0.946_{\pm 0.003}$	$0.973_{\pm 0.002}$	$0.958_{\pm 0.004}$	$0.970_{\pm 0.002}$	$0.976_{\pm 0.002}$	$\mathbf{0.900}_{\pm 0.003}$
	0.05	$0.947_{\pm 0.004}$	$0.985_{\pm 0.002}$	$0.919_{\pm 0.004}$	$0.955_{\pm 0.003}$	$0.938_{\pm 0.004}$	$0.950_{\pm 0.003}$	$0.965_{\pm 0.002}$	$\mathbf{0.835}_{\pm 0.004}$
GRB-Cora	0.01	$0.975_{\pm 0.003}$	$0.993_{\pm 0.001}$	$0.967_{\pm 0.002}$	$0.976_{\pm 0.002}$	$0.969_{\pm 0.002}$	$0.975_{\pm 0.001}$	$0.991_{\pm 0.006}$	$\mathbf{0.955}_{\pm 0.001}$
	0.03	$0.910_{\pm 0.003}$	$0.945_{\pm 0.003}$	$0.904_{\pm 0.003}$	$0.925_{\pm 0.004}$	$0.908_{\pm 0.004}$	$0.925_{\pm 0.004}$	$0.936_{\pm 0.010}$	$\mathbf{0.848}_{\pm 0.003}$
	0.05	$0.874_{\pm 0.005}$	$0.929_{\pm 0.003}$	$0.849_{\pm 0.006}$	$0.875_{\pm 0.005}$	$0.860_{\pm 0.003}$	$0.877_{\pm 0.006}$	$0.897_{\pm 0.024}$	$\mathbf{0.765}_{\pm 0.006}$
GRB-Citeseer	0.01	$0.981_{\pm 0.004}$	$0.995_{\pm 0.001}$	$0.977_{\pm 0.002}$	$0.982_{\pm 0.001}$	$0.968_{\pm 0.004}$	$0.973_{\pm 0.002}$	$0.982_{\pm 0.002}$	$\mathbf{0.957}_{\pm 0.002}$
	0.03	$0.937_{\pm 0.007}$	$0.966_{\pm 0.006}$	$0.925_{\pm 0.008}$	$0.943_{\pm 0.003}$	$0.914_{\pm 0.010}$	$0.918_{\pm 0.007}$	$0.953_{\pm 0.011}$	$\mathbf{0.876}_{\pm 0.004}$
	0.05	$0.891_{\pm 0.015}$	$0.930_{\pm 0.009}$	$0.883_{\pm 0.019}$	$0.900_{\pm 0.006}$	$0.862_{\pm 0.019}$	$0.865_{\pm 0.015}$	$0.927_{\pm 0.010}$	$\mathbf{0.797}_{\pm 0.005}$
Pubmed	0.01	$0.990_{\pm 0.000}$	$0.994_{\pm 0.000}$	$0.984_{\pm 0.001}$	$0.988_{\pm 0.000}$	$0.982_{\pm 0.001}$	$0.990_{\pm 0.000}$	$0.986_{\pm 0.007}$	$\mathbf{0.955}_{\pm 0.002}$
	0.03	$0.970_{\pm 0.000}$	$0.981_{\pm 0.001}$	$0.955_{\pm 0.001}$	$0.964_{\pm 0.001}$	$0.951_{\pm 0.003}$	$0.969_{\pm 0.001}$	$0.979_{\pm 0.006}$	$\mathbf{0.879}_{\pm 0.005}$
	0.05	$0.950_{\pm 0.001}$	$0.965_{\pm 0.001}$	$0.927_{\pm 0.002}$	$0.938_{\pm 0.002}$	$0.924_{\pm 0.005}$	$0.951_{\pm 0.002}$	$0.974_{\pm 0.010}$	$\mathbf{0.820}_{\pm 0.007}$
Arxiv	0.01	$0.983_{\pm 0.002}$	$0.996_{\pm 0.000}$	$0.978_{\pm 0.001}$	$0.982_{\pm 0.001}$	$0.972_{\pm 0.001}$	$0.977_{\pm 0.001}$	$0.982_{\pm 0.006}$	$\mathbf{0.943}_{\pm 0.001}$
	0.03	$0.943_{\pm 0.005}$	$0.971_{\pm 0.003}$	$0.937_{\pm 0.003}$	$0.943_{\pm 0.003}$	$0.926_{\pm 0.005}$	$0.934_{\pm 0.004}$	$0.933_{\pm 0.014}$	$\mathbf{0.852}_{\pm 0.002}$
	0.05	$0.900_{\pm 0.007}$	$0.934_{\pm 0.003}$	$0.893_{\pm 0.007}$	$0.907_{\pm 0.003}$	$0.881_{\pm 0.006}$	$0.889_{\pm 0.005}$	$0.903_{\pm 0.018}$	$\mathbf{0.778}_{\pm 0.003}$

Table 1: Average performance of various attack methods on undefended models, where lower values indicate better performance.

Datasets	a	TDGIA	TDGIA+HAO	ATDGIA	ATDGIA + HAO	AGIA	AIGA+HAO	G2A2C	QUAGIA
Cora	0.01	$0.998_{\pm 0.000}$	$0.992_{\pm 0.003}$	$0.996_{\pm 0.001}$	$0.982_{\pm 0.002}$	$0.993_{\pm 0.002}$	$0.980_{\pm 0.002}$	$0.992_{\pm 0.005}$	$\mathbf{0.956}_{\pm 0.004}$
	0.03	$0.991_{\pm 0.003}$	$0.965_{\pm 0.002}$	$0.987_{\pm 0.003}$	$0.955_{\pm 0.005}$	$0.985_{\pm 0.004}$	$0.938_{\pm 0.007}$	$0.992_{\pm 0.009}$	$\mathbf{0.880}_{\pm 0.005}$
	0.05	$0.984_{\pm 0.004}$	$0.909_{\pm 0.005}$	$0.983_{\pm 0.004}$	$0.908_{\pm 0.004}$	$0.981_{\pm 0.003}$	$0.910_{\pm 0.006}$	$0.982_{\pm 0.008}$	$\mathbf{0.818}_{\pm 0.004}$
Citeseer	0.01	$1.000_{\pm 0.000}$	$0.997_{\pm 0.000}$	$0.998_{\pm 0.000}$	$0.989_{\pm 0.001}$	$0.998_{\pm 0.000}$	$0.992_{\pm 0.001}$	$0.999_{\pm 0.000}$	$\mathbf{0.965}_{\pm 0.001}$
	0.03	$0.999_{\pm 0.000}$	$0.989_{\pm 0.001}$	$0.994_{\pm 0.001}$	$0.971_{\pm 0.002}$	$0.995_{\pm 0.000}$	$0.966_{\pm 0.003}$	$0.999_{\pm 0.000}$	$\mathbf{0.900}_{\pm 0.001}$
	0.05	$0.994_{\pm 0.003}$	$0.975_{\pm 0.002}$	$0.989_{\pm 0.003}$	$0.951_{\pm 0.003}$	$0.992_{\pm 0.002}$	$0.942_{\pm 0.003}$	$0.997_{\pm 0.002}$	$\mathbf{0.839}_{\pm 0.005}$
GRB-Cora	0.01	$0.997_{\pm 0.002}$	$0.988_{\pm 0.003}$	$0.990_{\pm 0.002}$	$0.964_{\pm 0.003}$	$0.995_{\pm 0.003}$	$0.962_{\pm 0.003}$	$0.995_{\pm 0.002}$	$\mathbf{0.956}_{\pm 0.003}$
	0.03	$0.992_{\pm 0.002}$	$0.920_{\pm 0.002}$	$0.989_{\pm 0.003}$	$0.889_{\pm 0.005}$	$0.993_{\pm 0.001}$	$0.890_{\pm 0.005}$	$0.924_{\pm 0.012}$	$\mathbf{0.831}_{\pm 0.006}$
	0.05	$0.989_{\pm 0.003}$	$0.902_{\pm 0.004}$	$0.985_{\pm 0.002}$	$0.819_{\pm 0.007}$	$0.987_{\pm 0.003}$	$0.828_{\pm 0.006}$	$0.904_{\pm 0.015}$	$\mathbf{0.742}_{\pm 0.005}$
GRB-Citeseer	0.01	$1.000_{\pm 0.000}$	$0.995_{\pm 0.000}$	$0.996_{\pm 0.000}$	$0.978_{\pm 0.002}$	$0.998_{\pm 0.000}$	$0.969_{\pm 0.004}$	$0.982_{\pm 0.006}$	$\mathbf{0.956}_{\pm 0.003}$
	0.03	$0.999_{\pm 0.000}$	$0.963_{\pm 0.005}$	$0.989_{\pm 0.001}$	$0.931_{\pm 0.008}$	$0.994_{\pm 0.001}$	$0.910_{\pm 0.013}$	$0.946_{\pm 0.008}$	$\mathbf{0.876}_{\pm 0.007}$
	0.05	$0.996_{\pm 0.002}$	$0.921_{\pm 0.008}$	$0.986_{\pm 0.001}$	$0.881_{\pm 0.009}$	$0.992_{\pm 0.001}$	$0.856_{\pm 0.017}$	$0.908_{\pm 0.015}$	$\mathbf{0.799}_{\pm 0.004}$
Pubmed	0.01	$0.999_{\pm 0.000}$	$0.988_{\pm 0.003}$	$0.997_{\pm 0.002}$	$0.985_{\pm 0.002}$	$0.998_{\pm 0.002}$	$0.980_{\pm 0.003}$	$0.986_{\pm 0.009}$	$\mathbf{0.961}_{\pm 0.001}$
	0.03	$0.995_{\pm 0.003}$	$0.971_{\pm 0.003}$	$0.994_{\pm 0.002}$	$0.953_{\pm 0.002}$	$0.996_{\pm 0.002}$	$0.953_{\pm 0.004}$	$0.964_{\pm 0.007}$	$\mathbf{0.899}_{\pm 0.002}$
	0.05	$0.996_{\pm 0.006}$	$0.951_{\pm 0.003}$	$0.989_{\pm 0.004}$	$0.925_{\pm 0.004}$	$0.994_{\pm 0.003}$	$0.930_{\pm 0.003}$	$0.935_{\pm 0.017}$	$\mathbf{0.833}_{\pm 0.004}$
Arxiv	0.01	$0.999_{\pm 0.000}$	$0.995_{\pm 0.000}$	$0.995_{\pm 0.001}$	$0.975_{\pm 0.001}$	$0.998_{\pm 0.000}$	$0.970_{\pm 0.002}$	$0.974_{\pm 0.012}$	$\mathbf{0.941}_{\pm 0.002}$
	0.03	$0.996_{\pm 0.000}$	$0.961_{\pm 0.004}$	$0.984_{\pm 0.002}$	$0.918_{\pm 0.004}$	$0.995_{\pm 0.000}$	$0.908_{\pm 0.007}$	$0.927_{\pm 0.014}$	$\mathbf{0.845}_{\pm 0.001}$
	0.05	$0.989_{\pm 0.002}$	$0.909_{\pm 0.004}$	$0.976_{\pm 0.003}$	$0.872_{\pm 0.007}$	$0.990_{\pm 0.001}$	$0.856_{\pm 0.007}$	$0.873_{\pm 0.021}$	$\mathbf{0.765}_{\pm 0.003}$

Table 2: Average performance of various attack methods on defended models, where lower values indicate better performance.

All attack methods were executed using five different random seeds, and we report both the mean and variance of the results across these five runs. The average performance of the attack methods is summarized in Table 1 and Table 2. In the no-defense setting, we present the average results across GCN, GAT, and APPNP, while in the defense setting, the results are averaged over Guard, EGuard, and RGAT. For additional details, including experimental configurations, comprehensive results, and sensitivity analyses, please refer to the *Supplementary Material*.

4.4 Performance Comparison

In Table 1 and Table 2, we present the test set classification accuracy of GNN models under various attack methods, with lower values indicating stronger attack performance. The best results are highlighted in bold, while the second-best results are underlined. These experimental results demonstrate that our attack method achieves superior performance on both defended and undefended GNN models. Specifically, our method outperforms AGIA by nearly 5% on average and up to 10% in the best case on undefended models. On defended models, our method performs significantly better on discrete datasets compared to other methods. This indicates that gradient-based approaches struggle with discrete data, as gradients are more suitable for optimizing features in continuous data. Although G2A2C reduces reliance on surrogate models, its focus on single-node attacks hinders the effective allocation of the attack budget across multiple nodes, leading

to weaker performance.

Additionally, we found that the performance of gradient-based attack methods drops significantly on defended models when the HAO module is removed. Our method achieves up to a 10% improvement compared to the effective AGIA+HAO attack method on defended models. Interestingly, attack methods with the HAO module generally perform worse on undefended models but better on defended ones. This suggests that the HAO module sacrifices some maliciousness to preserve homophily, making attacks less noticeable. Compared to HAO, our attack method implicitly preserves the homophily of the graph, making it harder to detect using homophily-based defense mechanisms. Furthermore, our attack method maintains excellent performance even on undefended GNN models.

Moreover, we observed that APPNP exhibits surprising robustness to GIA attacks, consistent with findings in previous work [Zhang *et al.*, 2023]. This robustness is likely due to the residual connections in APPNP, which preserve the original feature information. Our experiments show that different models exhibit varying levels of robustness to GIA attacks. Exploring effective GIA attack and defense mechanisms remains an important research direction.

4.5 Homophily Analysis

To further investigate the differences between our method and HAO, we compared our approach with the best baseline under defense models, TDGIA+HAO, on the Cora and GRB-Citeseer datasets with $\alpha = 0.03$ and a fixed seed of 0. Following the discussion in previous work [Chen *et al.*, 2022], we utilize node similarity to analyze the homophily distribution. Node similarity is defined as the similarity between the features of node u and the aggregated features of its neighbors. As illustrated in Figure 2, when TDGIA is applied without HAO, the node similarity of the perturbed graph deviates significantly from the distribution of the original clean graph. However, the addition of HAO effectively reduces this deviation. Furthermore, we observed that HAO is less effective on datasets with discrete features compared to those with continuous features, likely due to its reliance on surrogate model gradients.

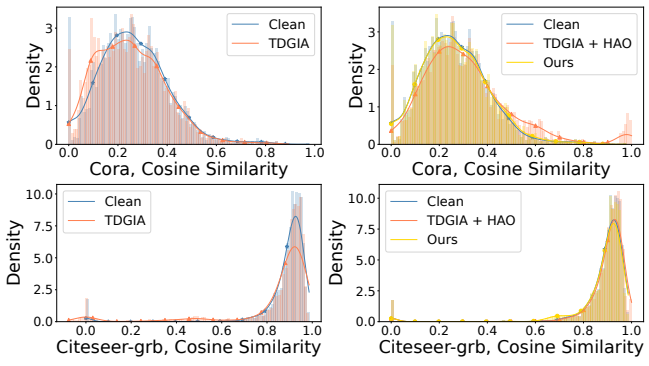

Figure 2: Node similarity distributions under different attacks for the Cora and GRB-Citeseer datasets. The x-axis represents cosine similarity, while the y-axis depicts the density of each similarity value.

Similar to HAO, our method also brings the homophily distribution of the perturbed graph closer to that of the original graph. Our attack method preserves the homophily of both discrete and continuous features. In GRB-Citeseer, we shifted a small number of nodes with a similarity of around 0.9 to approximately 0.7. Experimental results suggest that this shift is less likely to be detected by homophily defenders. Balancing the protection of homogeneity while enhancing the performance of attack methods against undefended models remains a challenge.

4.6 Ablation Analysis

Refer to the *Supplementary Material* for details on the ablation analysis. Detailed experimental results for Table 1 and Table 2 are also provided in the *Supplementary Material*.

5 Related Work

With the widespread adoption of GNNs, their robustness against adversarial attacks has gained increasing attention. While early studies mainly focused on GMA, constraints such as user permissions have led to a shift toward GIA, which has emerged as a more practical and effective attack approach. NIPA [Sun *et al.*, 2020] achieves its malicious intent by generating a batch of random nodes and injecting them into the existing graph. AFGSM [Wang *et al.*, 2020] employs the fast gradient sign method to inject malicious nodes, enabling attacks on large-scale graphs. G-NIA [Tao *et al.*, 2021] utilizes neural networks to learn and generate new nodes and edges, which are then injected into the original graph to launch attacks. TDGIA [Zou *et al.*, 2021] proposes a method for detecting vulnerable nodes in the graph topology to identify attack targets and introduces a smooth feature generation approach to facilitate attacks. AGIA [Chen *et al.*, 2022] utilizes surrogate model gradients to optimize edge weights and inject node features. HAO [Chen *et al.*, 2022] offers a plug-in method to enhance the unnoticeability of attacks, utilizing the homophily ratio of nodes as an unnoticeability constraint to further improve the stealthiness of adversarial attacks. However, the aforementioned methods all rely on surrogate models, and discrepancies between the target model and the surrogate model may lead to degraded attack performance. G2A2C [Ju *et al.*, 2023] formulates the attack process as a Markov Decision Process (MDP) and utilizes reinforcement learning to optimize an injection model based on query-based learning. While G2A2C alleviates dependency on surrogate model gradients, it does not consider the unnoticeability of the attack and still depends on surrogate models.

6 Conclusion

In this work, we propose a neighbor perspective-based attack method that uses a Bayesian framework to generate features for injected nodes, completely eliminating the dependence on surrogate models. Our attack method implicitly preserves the homophily of the original graph, ensuring excellent attack performance under both homophily defenders and undefended models. Extensive Experimental results show that our method outperforms existing state-of-the-art attack methods.

References

[Chen *et al.*, 2022] Yongqiang Chen, Han Yang, Yonggang Zhang, MA KAILI, Tongliang Liu, Bo Han, and James Cheng. Understanding and improving graph injection attack by promoting unnoticeability. In *International Conference on Learning Representations*, 2022.

[Croce and Hein, 2019] Francesco Croce and Matthias Hein. Sparse and imperceivable adversarial attacks. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4724–4732, 2019.

[Dai *et al.*, 2018] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial attack on graph structured data. In *International conference on machine learning*, pages 1115–1124. PMLR, 2018.

[Gasteiger *et al.*, 2019] Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Combining neural networks with personalized pagerank for classification on graphs. In *International Conference on Learning Representations*, 2019.

[Hu *et al.*, 2020] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

[Jiang *et al.*, 2021] Dejun Jiang, Zhenxing Wu, Chang-Yu Hsieh, Guangyong Chen, Ben Liao, Zhe Wang, Chao Shen, Dongsheng Cao, Jian Wu, and Tingjun Hou. Could graph neural networks learn better molecular representation for drug discovery? a comparison study of descriptor-based and graph-based models. *Journal of cheminformatics*, 13:1–23, 2021.

[Ju *et al.*, 2023] Mingxuan Ju, Yujie Fan, Chuxu Zhang, and Yanfang Ye. Let graph be the go board: gradient-free node injection attack for graph neural networks via reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 4383–4390, 2023.

[Kipf and Welling, 2017] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

[Lee *et al.*, 2022] Deokjae Lee, Seungyong Moon, Junhyeok Lee, and Hyun Oh Song. Query-efficient and scalable black-box adversarial attacks on discrete sequential data via bayesian optimization. In *International Conference on Machine Learning*, pages 12478–12497. PMLR, 2022.

[Li *et al.*, 2023] Kuan Li, Yang Liu, Xiang Ao, and Qing He. Revisiting graph adversarial attack and defense from a data distribution perspective. In *The Eleventh International Conference on Learning Representations*, 2023.

[Narodytska and Kasiviswanathan, 2017] Nina Narodytska and Shiva Prasad Kasiviswanathan. Simple black-box adversarial attacks on deep neural networks. In *CVPR Workshops*, volume 2, 2017.

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

[Sun *et al.*, 2020] Yiwei Sun, Suhang Wang, Xianfeng Tang, Tsung-Yu Hsieh, and Vasant Honavar. Adversarial attacks on graph neural networks via node injections: A hierarchical reinforcement learning approach. In *Proceedings of the Web Conference 2020*, pages 673–683, 2020.

[Tao *et al.*, 2021] Shuchang Tao, Qi Cao, Huawei Shen, Junjie Huang, Yunfan Wu, and Xueqi Cheng. Single node injection attack against graph neural networks. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 1794–1803, 2021.

[Veličković *et al.*, 2018] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[Vo *et al.*, 2024] Quoc Viet Vo, Ehsan Abbasnejad, and Damith Ranasinghe. BRUSLEATTACK: A QUERY-EFFICIENT SCORE- BASED BLACK-BOX SPARSE ADVERSARIAL ATTACK. In *The Twelfth International Conference on Learning Representations*, 2024.

[Wang *et al.*, 2020] Jihong Wang, Minnan Luo, Fnu Suya, Jundong Li, Zijiang Yang, and Qinghua Zheng. Scalable attack on graph data by injecting vicious nodes. *Data Mining and Knowledge Discovery*, 34:1363–1389, 2020.

[Wang *et al.*, 2021] Zhengyi Wang, Zhongkai Hao, Ziqiao Wang, Hang Su, and Jun Zhu. Cluster attack: Query-based adversarial attacks on graphs with graph-dependent priors. *arXiv preprint arXiv:2109.13069*, 2021.

[Xing *et al.*, 2024] Yujie Xing, Xiao Wang, Yibo Li, Hai Huang, and Chuan Shi. Less is more: on the over-globalizing problem in graph transformers. In *International Conference on Machine Learning*, pages 54656–54672. PMLR, 2024.

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

[Yang *et al.*, 2016] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pages 40–48. PMLR, 2016.

[Zhang and Zitnik, 2020] Xiang Zhang and Marinka Zitnik. Gnnguard: Defending graph neural networks against adversarial attacks. *Advances in neural information processing systems*, 33:9263–9275, 2020.

[Zhang *et al.*, 2023] Mengmei Zhang, Xiao Wang, Chuan Shi, Lingjuan Lyu, Tianchi Yang, and Junping Du. Minimum topology attacks for graph neural networks. In *Proceedings of the ACM Web Conference 2023*, pages 630–640, 2023.

[Zheng *et al.*, 2021] Qinkai Zheng, Xu Zou, Yuxiao Dong, Yukuo Cen, Da Yin, Jiarong Xu, Yang Yang, and Jie Tang.

Graph robustness benchmark: Benchmarking the adversarial robustness of graph machine learning. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[Zou *et al.*, 2021] Xu Zou, Qinkai Zheng, Yuxiao Dong, Xinyu Guan, Evgeny Kharlamov, Jialiang Lu, and Jie Tang. Tdgia: Effective injection attacks on graph neural networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 2461–2471, 2021.

[Zügner *et al.*, 2018] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2847–2856, 2018.

[Zügner and Günnemann, 2019] Daniel Zügner and Stephan Günnemann. Adversarial attacks on graph neural networks via meta learning. In *International Conference on Learning Representations*, 2019.