GHAttack: Generative Adversarial Attacks on Heterogeneous Graph Neural Networks

Shaoxin Li, Xiaofeng Liao, *Fellow, IEEE*, Huanzhang Zhu, Junqing Le and Lingyang Chu

***Abstract*—Heterogeneous graph neural networks (HGNNs) have witnessed remarkable progress and widespread applications in recent years. Meanwhile, there is growing attention regarding their vulnerability to adversarial attacks. Existing attack methods for HGNNs generate perturbations to slightly modify the structure of a heterogeneous graph, thereby degrading the predictive performance of HGNNs on target nodes. However, to craft such a perturbation, these methods require solving a complicated optimization problem, which makes them computationally inefficient for launching attacks during the inference phase. In this work, we therefore introduce GHAttack, a novel generative attack method for efficient and effective adversarial attacks on HGNNs. Specifically, GHAttack aims to train a perturbation generator, which produces a perturbation for each target node via a simple forward pass, while allowing the perturbation to modify edges on the heterogeneous relations of the graph to obtain high attack effectiveness. To achieve this, we design a novel model architecture for the generator, consisting of an HGNN backbone and a relation-aware output layer. We formulate the training of the generator as an optimization problem and efficiently solve it by addressing a series of technical challenges. Extensive experiments on ten representative HGNNs and six datasets verify the high efficiency and excellent effectiveness of GHAttack.**

***Index Terms*—Heterogeneous graph neural network, adversarial attack, generative model, structure-based attack**

I. Introduction

HETEROGENEOUS graph neural networks (HGNNs) have attracted considerable research interest due to their ability to handle heterogeneous graphs that consist of multiple types of nodes and edges/relations. Through specialized learning processes, HGNNs capture rich semantics and complex structure information from the *heterogeneous relations* of the graphs, thus enabling the extraction of informative node representations for graph mining [1]–[6]. In recent years, a variety of HGNNs have been developed and have shown excellent performance on a series of tasks, such as node classification [7], link prediction [8], and recommendation [9].

Despite the remarkable success of HGNNs, their broad application in many critical areas, such as e-commerce [10] and cybersecurity [11], has also drawn growing attention to their vulnerability against *adversarial attacks*. A few recent studies [12]–[14] have revealed that a well-trained HGNN for node classification is susceptible to adversarial *perturbations*, which slightly modify the structure of a heterogeneous graph (e.g., adding or deleting edges) to mislead the HGNN to make incorrect predictions for target nodes in the graph (as shown in Fig. 1).

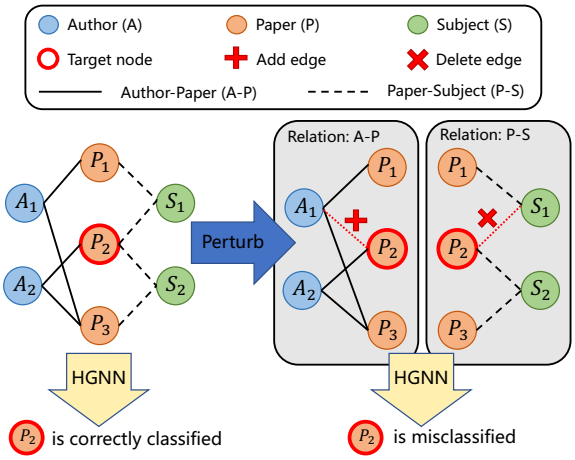

Fig. 1. An adversarial attack against an HGNN, which perturbs the structure of a heterogeneous graph. (Left) The original heterogeneous graph with three node types { "Author (A)", "Paper (P)", "Subject (S)"} and two edge types/relations {"Author-Paper (A-P)", "Paper-Subject (P-S)"}. (Right) Adding and deleting edges on the heterogeneous relations A-P and P-S.

While effective, the attack methods proposed in [12]–[14] are not time-efficient, as they require generating a perturbation for each target node by solving a complicated optimization problem. Moreover, since every perturbation is target-specific, the perturbation for each target node must be independently optimized from scratch. Consequently, it is impractical for these methods to launch rapid and massive attacks during the inference phase [15]. For example, HGAttack [13] takes over 40 hours to generate perturbations for 13,000 target nodes on the DBLP dataset [1].

In this work, we therefore study a novel problem of **efficient and effective adversarial attacks** on HGNNs, where the goal is to launch fast attacks while achieving a high attack success rate. To the best of our knowledge, this is a new yet challenging problem that has not been well investigated in the literature. As to be discussed in Section II, existing *optimization-based methods* [12]–[14], [16]–[19] cannot efficiently launch an attack for HGNNs due to the expensive time cost of solving an optimization problem to generate a perturbation. Furthermore, existing *model-based methods* [20]–[24],

This work was done by Shaoxin Li during his visit at McMaster University when supervised by Lingyang Chu. This work is supported in part by the NSERC Discovery Grant program (RGPIN-2022-04977), in part by National Natural Science Foundation of China (Grant no. 62202071), in part by the Natural Science Foundation of Chongqing (Innovation and Development Joint Fund) under grant CSTB2023NSCQ-LZX0149, in part by the Fundamental Research Funds for the Central Universities under grant 2023CDJKYJH019, and in part by the scholarship from China Scholarship Council. (*Corresponding author: Xiaofeng Liao*)

S. Li, X. Liao and J. Le are with the College of Computer Science, Chongqing University, Chongqing, 400044, China (email: {shaoxin.li, junqingle, xfliao}@cqu.edu.cn).

H. Zhu and L. Chu are with the Department of Computing and Software, McMaster University, Hamilton, L8S 4L8, Canada (email: {zhuh98, chul9}@mcmaster.ca).

which are designed for attacking classic graph neural networks (GNNs) [25] on homogeneous graphs, fail to achieve high attack effectiveness when applied to target HGNNs on heterogeneous graphs, as they overlook the graph heterogeneity and can only perturb edges on a single relation of the graph [12], [13], [26] (e.g., edges either on relation "A-P" or on relation "P-S" in Fig. 1).

To tackle this problem, we introduce a novel attack method called **generative heterogeneous attack (GHAttack)**. The *key idea* of GHAttack is to train a parametric model, named **perturbation generator**, such that 1) producing a perturbation for each target node only involves a forward pass through the generator, and 2) the produced perturbation modifies edges on heterogeneous relations to attain high attack effectiveness. Once trained, the generator is used during the inference phase to launch rapid and effective attacks against HGNNs for new target nodes.

To achieve this purpose, we first mathematically model a perturbation to represent edge modifications on heterogeneous relations. Then we design a novel model architecture for the generator by an *HGNN backbone* and a *relation-aware output layer*, such that its output characterizes the distribution of a perturbation. By adopting a CW-type loss function [26] to evaluate the effectiveness of perturbations, we formulate the training of the generator as an optimization problem, with the goal of maximizing the expected effectiveness of produced perturbations. Next, we propose a new method to solve the optimization problem by leveraging the *Gumbel softmax trick* [27]. In addition, we present *two strategies* to further improve efficiency by reducing the perturbation search space. We systematically evaluate the proposed GHAttack and compare it with five baseline methods on ten representative HGNNs and six benchmark datasets. Extensive experiments verify that GHAttack not only delivers efficient attacks by quickly generating perturbations for target nodes, but also demonstrates high attack effectiveness comparable to that of the best optimization-based method.

To sum up, the key contributions of this work are as follows.

- To our best knowledge, this is the first study on achieving both time-efficient and effective adversarial attacks against HGNNs. We propose a novel attack method named GHAttack, which successfully addresses the limitations of existing attack methods to achieve high attack effectiveness while maintaining high attack speed.
- GHAttack aims to train a perturbation generator that can swiftly produce a perturbation for each target node to modify edges on heterogeneous relations. To this end, we design a novel architecture for the perturbation generator, formulate its training as an optimization problem, and efficiently solve this problem by innovatively and effectively utilizing multiple techniques to tackle a series of technical challenges.
- We conduct extensive experiments on ten HGNNs and six benchmark datasets to demonstrate the excellent attack efficiency and great attack effectiveness of GHAttack.

II. Related Works

In this work, we investigate adversarial attacks on heterogeneous graph neural networks (HGNNs). We focus on *evasion attacks* that occur during the inference phase, where an attacker aims to mislead a well-trained HGNN with fixed parameters to make incorrect predictions on targets. Our work generally relates to two categories of attack methods for graph neural networks (GNNs), i.e., optimization-based methods and model-based methods.

Optimization-based methods. Given a target, such as a node, an edge or a graph, optimization-based methods formulate the attack as an optimization problem and solve it via typical techniques such as gradient descent to generate a target-specific perturbation. This perturbation is then used to perturb graph structure or node features for a successful attack. Such attack strategy has been proven to be effective against both classic GNNs on homogeneous graphs [16]–[19] and HGNNs on heterogeneous graphs [12]–[14], [26]. Nevertheless, since solving the optimization problem generally requires multiple iterations to update the perturbation, the time cost for a single attack can reach hundreds of seconds in the worst case [28], [29]. Consequently, a major limitation of the optimization-based methods lies in their inefficiency in launching rapid, large-scale attacks.

Model-based methods. Instead of crafting a perturbation through optimization, model-based methods [20]–[24] train a parametrized model from which a perturbation for a given target can be produced through simple computations (e.g., a forward pass). Once trained, the model can generalize to quickly generate perturbations for new targets, thus resulting in much faster attack speed than the optimization-based methods. Two main types of models have been studied, including the GNN family models [21]–[24] that are trained using either gradient-based methods or reinforcement learning, and the anchor node models [20] where the perturbation for a target node is inferred by flipping its connections to a set of anchor nodes.

Despite the fast attack speed, these model-based methods cannot handle graph heterogeneity since they are designed to target classic GNNs on homogeneous graphs that consist of a single type of edges [13], [26]. Therefore, when applied to attack HGNNs on a heterogeneous graph, they can only perturb edges on a single relation of the graph [12], [13], [26]. This limits their attack effectiveness, as HGNNs make predictions relying on messages passed by edges on heterogeneous relations rather than on a particular one.

Different from the above methods, we study a more challenging attack against HGNNs that pursues both high efficiency and high effectiveness. Our attack method, GHAttack, falls in the category of model-based methods by introducing a perturbation generator, thus eliminating the necessity of solving a time-consuming optimization problem to produce a perturbation. Furthermore, GHAttack bridges the research gap between model-based methods and adversarial attacks on HGNNs by enabling the generator to search for perturbations on heterogeneous relations. This offers great potential to achieve a higher attack success rate on HGNNs.

III. Preliminaries

A. Heterogeneous Graph

A heterogeneous graph, defined as $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, X, \mathcal{T}, \mathcal{R}\}$, consists of a set of nodes $\mathcal{V}$ and a set of edges $\mathcal{E}$. Each node $v \in \mathcal{V}$ is associated with a feature vector of real numbers that captures the meta information of each node. We denote by X the set of all node features in the heterogeneous graph $\mathcal{G}$. Each node v is also associated with a node type, where the set of all node types is denoted by $\mathcal{T} = \{T_1, \ldots, T_q\}$. Each edge $e \in \mathcal{E}$ is associated with an edge type, where the set of all edge types is denoted by $\mathcal{R} = \{R_1, \ldots, R_l\}$. Here, q and l are the number of node types and edge types, respectively. A classic heterogeneous graph $\mathcal{G}$ has $q + l > 2$ [8].

Example 1. *Fig. 1 illustrates an example of a classic heterogeneous graph, where the set of node types is* $\mathcal{T} =$ { *"Author", "Paper", "Subject"*} *and the set of edge types is* $\mathcal{R} =$ {*"Author-Paper", "Paper-Subject"*}. *For the simplicity of notations, we represent all node types by their initials, that are, "A", "P", and "S", respectively; and we write the edge types using abbreviations "A-P" and "P-S", correspondingly.*

In this work, we consider a typical kind of heterogeneous graph, where an edge type is determined by (i.e., matches with) the types of the two connected nodes [1], [30], [31]. In this case, an edge type is also referred to as a **relation** [12], [13], [26]. For instance, in Example 1, the relation "A-P" is determined by the node types "A" and "P". Formally, we define a relation as $R_i := T_a$–$T_b \in \mathcal{R}$, where $T_a \in \mathcal{T}$ and $T_b \in \mathcal{T}$ are the two associated node types.

Following the prior works [12], [13], [26], we treat a heterogeneous graph $\mathcal{G}$ with l relations as a set of l bipartite graphs. For instance, the heterogeneous graph in Fig. 1 is viewed as two bipartite graphs corresponding to the heterogeneous relations "A-P" and "P-S", respectively. Each bipartite graph, which corresponds to a relation $R_i := T_a$–$T_b \in \mathcal{R}$, consists of edges of a single type and describes the connectivity of nodes of types T_a and T_b. We represent its graph structure by a *binary adjacency matrix* $A_{R_i} \in \{0,1\}^{N_{T_a} \times N_{T_b}}$, where N_{T_a} and N_{T_b} are the number of nodes in the heterogeneous graph $\mathcal{G}$ with types T_a and T_b, respectively. Let $v^{(m)}$ be the m-th node of type T_a and $v^{(n)}$ be the n-th node of type T_b in $\mathcal{G}$. The entry $[A_{R_i}]_{m,n}$ at the m-th row and n-th column of A_{R_i} indicates the existence of an edge of type R_i between the nodes $v^{(m)}$ and $v^{(n)}$. Denoted by $\mathcal{A} = \{A_{R_1}, \ldots, A_{R_l}\}$ the set of binary adjacency matrices corresponding to the heterogeneous relations $R_1, \ldots, R_l$, it characterizes the overall graph structure of the heterogeneous graph $\mathcal{G}$.

B. Heterogeneous Graph Neural Network

Heterogeneous graph neural networks (HGNNs) are a subclass of graph neural networks specializing in handling heterogeneous graphs. A typical HGNN, denoted as f, generally takes a heterogeneous graph $\mathcal{G}$ as input, while its output varies depending on the specific tasks. In this work, we focus on the representative node classification task of HGNNs [30], [32]. Specifically, let $\mathcal{C} = \{1, 2, \ldots, C\}$ be a set of class labels, an HGNN f aims to predict the class label of a node v in the heterogeneous graph $\mathcal{G}$ by producing a C-dimensional vector of classification logits $\hat{y} \in \mathbb{R}^C$. This process is expressed as

$$\hat{y} = f(\mathcal{A}, X, v), \tag{1}$$

where $\mathcal{A}$ represents the graph structure of $\mathcal{G}$ and X is the set of node features. The predicted class label for v is computed by $\operatorname{argmax}_{j=1}^{C} \hat{y}_j$, where $\hat{y}_j$ is the j-th entry of the vector $\hat{y}$.

IV. Problem Definition

A. Threat Model

In the following, we present the threat model used in this paper, which includes three aspects.

Attackers' goal. We consider the *non-targeted evasion attack* against HGNNs. Concretely, given a well-trained HGNN f that performs node classification on a heterogeneous graph $\mathcal{G}$, an attacker aims to perturb $\mathcal{G}$ such that a target node v on the perturbed graph is misclassified by the victim HGNN f. An attack succeeds if the predicted class label of v on the perturbed graph is different from its ground-truth class label.

Attackers' prior knowledge. We consider both the *white-box* [23], [26] and the *gray-box* attack settings [12], [13]. In the white-box setting, we assume that the attacker has full access to the heterogeneous graph $\mathcal{G}$ and the labels of the training nodes of $\mathcal{G}$, as well as the architecture and parameters of the victim HGNN f. In the gray-box setting, we assume that the attacker only has access to the heterogeneous graph $\mathcal{G}$ and the training labels, lacking any knowledge about the victim HGNN f. This is also considered as the non-strict black-box setting in [26].

Attackers' capabilities. We assume that the attacker can only modify the graph structure of the heterogeneous graph $\mathcal{G}$ with a limited budget [13], [23], [26]. To attack the target node v, the attacker can add new edges or delete the existing edges in $\mathcal{G}$. To ensure that this attack is unnoticeable, we set a budget $\xi \in \mathbb{Z}^+$ to guarantee that the total number of modified edges is not larger than ξ.

B. Problem Statement

Given a victim HGNN f that performs node classification on a heterogeneous graph $\mathcal{G}$, a target node v in $\mathcal{G}$ and a predefined budget $\xi \in \mathbb{Z}^+$, the goal of **efficient and effective adversarial attacks** against HGNNs is to swiftly generate a perturbation to perturb $\mathcal{G}$ by adding and deleting at most ξ edges, such that the victim HGNN f will misclassify the target node v by producing an incorrect class label. More specifically, the perturbation modifies the graph structure of $\mathcal{G}$ that is represented by a set of binary adjacency matrices $\mathcal{A} = \{A_{R_1}, \ldots, A_{R_l}\}$, to a different graph structure represented by another set of binary adjacency matrices of the same sizes, denoted by $\tilde{\mathcal{A}} = \{\tilde{A}_{R_1}, \ldots, \tilde{A}_{R_l}\}$, to ensure $f(\tilde{\mathcal{A}}, X, v) \neq y$ where $y \in \mathcal{C}$ is the ground-truth class label of the target node v, while satisfying $\sum_{i=1}^{l} \|A_{R_i} - \tilde{A}_{R_i}\|_1 \leq \xi$.

V. Generative Heterogeneous Attack

In this paper, we propose a novel attack method, called generative heterogeneous attack (GHAttack), for efficient and

effective adversarial attacks on HGNNs. At its core, GHAttack trains a perturbation generator g_θ parameterized by θ, such that for a target node v, an effective perturbation representing limited edge modifications on a heterogeneous graph $\mathcal{G}$ can be swiftly produced via a forward pass through g_θ. In the following, we first introduce the modeling of such a perturbation and the design details of g_θ. Then, we formulate the training of g_θ as an optimization problem and efficiently solve it by tackling technical issues. Next, we present the training algorithm for g_θ and the details of launching an attack using the trained g_θ during the inference phase. Last, we analyze the time complexity of our attack method.

A. Modeling Perturbation

We first model a perturbation in order to describe mathematically the set of edge additions and deletions it involves. In particular, we aim to perform these edge modifications on the heterogeneous relations of the heterogeneous graph $\mathcal{G}$. To achieve this, for each relation $R_i := T_a$–$T_b \in \mathcal{R}$, we define a **binary perturbation matrix** $P_{R_i} \in \{0,1\}^{N_{T_a},\times N_{T_b}}$ of the same size as the binary adjacency matrix A_{R_i} to represent the edge modifications on the relation R_i. The m-th row and n-th column entry of P_{R_i}, denoted by $[P_{R_i}]_{m,n}$, indicates whether to flip the binary entry $[A_{R_i}]_{m,n}$. That is, if $[P_{R_i}]_{m,n} = 1$, then flip $[A_{R_i}]_{m,n}$ to add or delete the edge corresponding to $[A_{R_i}]_{m,n}$; otherwise, do not flip $[A_{R_i}]_{m,n}$ and keep the edge status unchanged. Let $\mathcal{P} = \{P_{R_1}, \ldots, P_{R_l}\}$ be the set of binary perturbation matrices for the l relations of $\mathcal{G}$, since it exactly defines a set of edge modifications on heterogeneous relations, we abuse the term of perturbation to refer to it.

Given a perturbation $\mathcal{P} = \{P_{R_1}, \ldots, P_{R_l}\}$, we leverage it to modify the graph structure of $\mathcal{G}$, which is represented by $\mathcal{A} = \{A_{R_1}, \ldots, A_{R_l}\}$, to a different graph structure denoted by $\tilde{\mathcal{A}} = \{\tilde{A}_{R_1}, \ldots, \tilde{A}_{R_l}\}$. Specifically, on each relation $R_i \in \mathcal{R}$, we have

$$\tilde{A}_{R_i} = A_{R_i} + \left(\overline{A_{R_i}} - A_{R_i}\right) \odot P_{R_i}, \tag{2}$$

where the complement matrix $\overline{A_{R_i}}$ is obtained by flipping all binary entries in A_{R_i} and $\odot$ denotes the Hadamard product between two matrices. For simplicity, we introduce a *perturbing function* Q to summarize the above process for all the relations in $\mathcal{R}$, which is stated as

$$\tilde{\mathcal{A}} = Q\left(\mathcal{A}, \mathcal{P}\right). \tag{3}$$

B. Designing Perturbation Generator

In this subsection, we detail the design of the perturbation generator g_θ that is used in GHAttack to swiftly generate a perturbation $\mathcal{P}$ for a target node v. Since $\mathcal{P}$ consists of multiple binary perturbation matrices on different relations, it requires g_θ to predict the values of entries in each of these perturbation matrices. To this end, we implement g_θ as consisting of an HGNN backbone and a relation-aware output layer, as described below.

HGNN backbone. To obtain useful information for predicting the perturbation $\mathcal{P}$, we employ an HGNN backbone to extract node embedding for all nodes in the heterogeneous graph $\mathcal{G}$. Formally, denoted by h_η an HGNN backbone parameterized by η, we express the extraction process as

$$z^{(1)}, \ldots, z^{(|\mathcal{V}|)} = h_\eta(\mathcal{A}, X), \tag{4}$$

where $|\mathcal{V}|$ is the number of nodes in $\mathcal{G}$ and $z^{(j)} \in \mathbb{R}^H$ is the H-dimensional embedding for the j-th node $v^{(j)}$ of $\mathcal{G}$.

Relation-aware output layer. Following the HGNN backbone h_η, the output layer of the generator g_θ leverages the information encoded in the node embeddings to predict the values of entries in the perturbation $\mathcal{P}$. Recall that different perturbation matrices of $\mathcal{P}$ are associated with different relations, and each of them indicates the edge modifications on a particular relation. Therefore, when predicting the values of entries in each perturbation matrix, we propose to leverage relation-specific information encoded in the node embeddings to provide more accurate predictions.

Concretely, for an entry $[P_{R_i}]_{m,n}$ in the perturbation matrix P_{R_i} associated with the relation $R_i := T_a$–$T_b \in \mathcal{R}$, denoted by $v^{(m)}$ the m-th node of type T_a and $v^{(n)}$ the n-th node of type T_b, we further process their node embeddings to acquire the information specifically related to R_i. Taking the node $v^{(m)}$ as an example, we convert its node embedding $z^{(m)}$ to the *relation-specific embedding* $z_{R_i}^{(m)}$ as follows:

$$z_{R_i}^{(m)} = \sum_{c \in \mathcal{N}_{R_i}} \frac{1}{|\mathcal{N}_{R_i}|} W_{R_i} z^{(c)} + W_0 z^{(m)}, \tag{5}$$

where $\mathcal{N}_{R_i}$ is the index set of the neighboring nodes connected to $v^{(m)}$ through the relation R_i, and $W_{R_i} \in \mathbb{R}^{H \times H}$ and $W_0 \in \mathbb{R}^{H \times H}$ are trainable weight matrices. Next, instead of predicting the binary value of $[P_{R_i}]_{m,n}$, we use the relation-specific embeddings to estimate the probability that the value of $[P_{R_i}]_{m,n}$ equals to one. This circumvents the issue of non-differentiability caused by predicting binary values through quantization. Denote this probability as $\phi_{R_i}^{(m,n)}$, it is computed by

$$\phi_{R_i}^{(m,n)} = \sigma\left(M_1\left(M_0\left(z_{R_i}^{(m)} + z_{R_i}^{(n)}\right)\right)\right), \tag{6}$$

where $z_{R_i}^{(m)}$ and $z_{R_i}^{(n)}$ are, respectively, the relation-specific embeddings of the nodes $v^{(m)}$ and $v^{(n)}$ with respect to the relation R_i. $M_0 \in \mathbb{R}^{H/2 \times H}$ and $M_1 \in \mathbb{R}^{1 \times H/2}$ are trainable weight matrices, and $\sigma(\cdot)$ is the sigmoid function.

Built upon the HGNN backbone h_η and the relation-aware output layer, the generator g_θ outputs a set of probabilities for the target node v, denoted as $\Phi = g_\theta(\mathcal{A}, X, v)$. By treating each binary entry $[P_{R_i}]_{m,n}$ as a Bernoulli random variable and $\mathbb{P}([P_{R_i}]_{m,n} = 1) = \phi_{R_i}^{(m,n)}$, the set of probabilities Φ collectively characterizes a multivariate Bernoulli distribution over the perturbation $\mathcal{P}$. Thus, we have

$$\mathcal{P} \sim \mathrm{Ber}(\Phi) = \mathrm{Ber}(g_\theta(\mathcal{A}, X, v)), \tag{7}$$

where $\theta = \{\eta, W_{R_1}, \ldots, W_{R_l}, W_0, M_1, M_0\}$.

C. Formulating the Training

In this subsection, we formulate the training of the generator g_θ as an optimization problem, such that a perturbation $\mathcal{P}$ sampled from the output of g_θ can effectively attack the victim

HGNN f for the target node v. We start by considering the white-box setting where f is known to the attacker, and then discuss the formulation in the gray-box setting where f is unknown to the attacker.

First, we introduce a CW-type loss function [26] to evaluate the attack effectiveness of a perturbation $\mathcal{P}$. Let $\tilde{\mathcal{A}}$ represent the graph structure perturbed by $\mathcal{P}$ according to (3), the output of f for the target node v on the perturbed graph $\tilde{\mathcal{A}}$ is a vector of classification logits $\hat{y} = f(\tilde{\mathcal{A}}, X, v)$, where the j-th entry in the vector $\hat{y}$, denoted by $\hat{y}_j$, is the logit value for the j-th class. Then, the attack effectiveness of the perturbation $\mathcal{P}$ is evaluate by

$$L(\hat{y}, y) = \max\big(\hat{y}_y - \max_{j \neq y} \hat{y}_j, -\kappa\big), \tag{8}$$

where $y \in \mathcal{C}$ is the ground-truth class label of the target node v and $\kappa \geq 0$ is a real-valued parameter used to control the degree of misclassification. Here, $\hat{y}_y$ is the logit value for the ground-truth class label and $\max_{j \neq y} \hat{y}_j$ is the largest logit value among the other classes. A smaller loss value of $L(\hat{y}, y)$ means a higher degree of misclassification, implying higher attack effectiveness of $\mathcal{P}$. A larger value of the parameter κ allows for a higher degree of misclassification.

Next, we formulate the training of the generator g_θ in the white-box setting as the following optimization problem:

$$\begin{aligned} \min_{\theta} \quad & \mathbb{E}_{v \sim \mathcal{V}_L} \mathbb{E}_{\mathcal{P} \sim \mathrm{Ber}(g_\theta(\mathcal{A}, X, v))} \big[L\big(f(Q(\mathcal{A}, \mathcal{P}), X, v, y\big) \\ & + \lambda \max\Big(\sum_{i=1}^{l} \|P_{R_i}\|_1 - \xi, 0\Big)\big], \end{aligned} \tag{9}$$

where $\mathcal{V}_L$ is the empirical distribution specified by a set of labeled nodes in the heterogeneous graph $\mathcal{G}$ and $\lambda > 0$ is a penalty parameter for violations of the budget constraint $\sum_{i=1}^{l} \|P_{R_i}\|_1 \leq \xi$. Minimizing the objective function in (9) essentially maximizes the attack effectiveness of the perturbations sampled with high probability from the distribution $\mathrm{Ber}(g_\theta(\mathcal{A}, X, v))$ while encouraging these perturbations not to violate the budget constraint. Thus, the trained generator g_θ is expected to generalize to produce effective perturbations for new target nodes during the inference phase using the same budget. In practice, we gradually increase the penalty parameter λ during training so that the perturbations sampled from the trained g_θ are likely to adhere to the budget ξ.

In the gray-box setting, the victim HGNN f in (9) is not accessible to the attacker. Instead, the attacker can use a surrogate HGNN (denoted as f_S) to substitute f and solve the same problem in (9) to train g_θ. The surrogate HGNN f_S adopts a different model architecture than f, but it is also trained to perform node classification on the heterogeneous graph $\mathcal{G}$. The key idea is that, if a perturbation produced by the trained g_θ can effectively attack f_S, then it may also be able to successfully attack f due to the high task relevance between f_S and f. This is also well-known as transfer adversarial attacks in the literature [14], [15], [26].

D. Solving the Optimization Problem

Solving the optimization problem in (9) needs to explicitly compute the expectation $\mathbb{E}_{\mathcal{P} \sim \mathrm{Ber}(g_\theta(\mathcal{A}, X, v))}[\cdot]$ for every target node v. However, this could be computationally expensive as it involves computing each probability in Φ by (6) and the loss term $L(f(Q(\mathcal{A}, \mathcal{P}), X, v), y) + \lambda \max(\sum_{i=1}^{l} \|P_{R_i}\|_1 - \xi, 0)$ for each possible perturbation. Denote by k the maximum time cost of computing each probability in Φ and by w the maximum time cost of computing the loss term for each possible perturbation. Since the maximum number of entries in Φ is equal to the maximum number of possible edges in the heterogeneous graph $\mathcal{G}$, which is $|\mathcal{V}|(|\mathcal{V}|-1)/2$, and the maximum number of possible perturbations is $2^{|\mathcal{V}|(|\mathcal{V}|-1)/2}$, the worst-case time cost of computing $\mathbb{E}_{\mathcal{P} \sim \mathrm{Ber}(g_\theta(\mathcal{A}, X, v))}[\cdot]$ is $k|\mathcal{V}|(|\mathcal{V}|-1)/2 + w2^{|\mathcal{V}|(|\mathcal{V}|-1)/2} \sim O\big(k|\mathcal{V}|^2 + w2^{|\mathcal{V}|^2}\big)$. Consequently, the doubly exponential growth makes the exact computation of this expectation infeasible for large $|\mathcal{V}|$.

To tackle this issue, we propose a method to reduce the time complexity of computing the expectation $\mathbb{E}_{\mathcal{P} \sim \mathrm{Ber}(g_\theta(\mathcal{A}, X, v))}[\cdot]$ to be linear in $|\mathcal{V}|$. This is accomplished by first reducing the perturbation search space and then applying the Gumbel softmax trick [27], which is described as follows.

Reducing the perturbation search space. Computing the probabilities in Φ for at most $|\mathcal{V}|(|\mathcal{V}|-1)/2$ entries means deciding whether to perturb each of the possible edges on $\mathcal{G}$, which constitutes a perturbation search space of size $|\mathcal{V}|(|\mathcal{V}|-1)/2$. As a result, if $|\mathcal{V}|$ is very large, the huge perturbation search space would make the computation expensive.

To address this issue, we propose to reduce the perturbation search space by selecting a small subset of possible edges as candidate edges for perturbing and only computing the probabilities in Φ corresponding to the selected candidate edges. We present two strategies to select the candidate edges that may have a significant influence on classifying the target node v. *Strategy 1:* any existing edge within the subgraph induced by the ϵ-hop ($\epsilon \in \mathbb{Z}_{\geq 0}$) neighborhood of v is selected as a candidate edge for deletion. *Strategy 2:* for all non-existing edges that can connect v with another node $v_o \in \mathcal{V} \setminus v$ on a valid relation, we uniformly sample a proportion $\beta \in [0, 1]$ of them at random as the candidate edges for addition.

Strategy 1 allows us to predict the deletion of an existing edge within the ϵ-hop subgraph of the target node v. Denoted by $|\mathcal{E}|$ the number of edges in $\mathcal{G}$. The maximum size of the perturbation search space derived from this strategy is $|\mathcal{E}|$ if ϵ is large enough such that the ϵ-hop subgraph is the entire graph. Strategy 2 allows us to predict the addition of a non-existing edge selected with probability β that can connect v to another node in $\mathcal{G}$ through a valid relation. We do not consider adding non-existing edges that are not directly connected to v, as it has been demonstrated to be less effective for attacks [33]. The maximum size of the perturbation search space derived from this strategy is $|\mathcal{V}| - 1$ if $\beta = 1$. Thus, the proposed strategies reduce the maximum size of the perturbation search space from $|\mathcal{V}|(|\mathcal{V}|-1)/2$ to $|\mathcal{V}| + |\mathcal{E}| - 1$. Since $|\mathcal{E}|$ is often larger than $|\mathcal{V}|$ by a constant factor due to the high sparsity of real-world graphs, we have $|\mathcal{E}| \sim O(|\mathcal{V}|)$. Therefore, the time complexity of computing $\mathbb{E}_{\mathcal{P} \sim \mathrm{Ber}(g_\theta(\mathcal{A}, X, v))}[\cdot]$ is reduced to $k(|\mathcal{V}| + |\mathcal{E}| - 1) + w2^{|\mathcal{V}|+|\mathcal{E}|-1} \sim O\big(k|\mathcal{V}| + w2^{|\mathcal{V}|}\big)$. For the existing or non-existing edges that are not selected as the candidate edges by the proposed two strategies, we do not

compute the probabilities in Φ that correspond to them and simply set these probabilities to zeros.

Applying the Gumbel softmax trick. Given the reduced perturbation search space, we apply the Gumbel softmax trick to further reduce the time complexity of computing the expectation $\mathbb{E}_{\mathcal{P}\sim\mathrm{Ber}(g_\theta(\mathcal{A},X,v))}[\cdot]$. Essentially, the Gumbel softmax trick is a differentiable approximate sampling mechanism, which allows us to estimate $\mathbb{E}_{\mathcal{P}\sim\mathrm{Ber}(g_\theta(\mathcal{A},X,v))}[\cdot]$ by the Monte Carlo method [34] while preserving the gradients with respect to θ. We apply it by the following two steps.

First, we approximate the discrete perturbation $\mathcal{P}$ in (9) by a smooth perturbation of the same size as $\mathcal{P}$, denoted by $\mathcal{S}$. This approximates each entry $[P_{R_i}]_{m,n}$ in $\mathcal{P}$ by

$$[S_{R_i}]_{m,n} = \sigma\left(\frac{\log\alpha + \log[U_{R_i}]_{m,n} - \log(1-[U_{R_i}]_{m,n})}{\tau}\right), \tag{10}$$

where $\alpha = \phi_{R_i}^{(m,n)}/(1-\phi_{R_i}^{(m,n)})$, $[U_{R_i}]_{m,n} \sim \mathrm{Unif}(0,1)$ is a uniform random variable and $\tau > 0$ is a temperature parameter. A smaller value of τ makes the entries in $\mathcal{S}$ get closer to 0 or 1.

Second, we convert the optimization problem in (9) to

$$\min_{\theta} \quad \mathbb{E}_{v\sim\mathcal{V}_L}\mathbb{E}_{\mathcal{U}\sim\mathrm{Unif}(0,1)}\big[L\big(f\left(Q\left(\mathcal{A},\mathcal{S}\right),X,v\right),y\big) + \lambda\max\big(\sum_{i=1}^{l}\|S_{R_i}\|_1 - \xi, 0\big)\big], \tag{11}$$

where $\mathcal{U}$ has the same size as $\mathcal{S}$ and each of its entries $[U_{R_i}]_{m,n}$ is an independent random variable following the uniform distribution $\mathrm{Unif}(0,1)$.

By applying the above Gumbel softmax trick, we can approximate the expectation $\mathbb{E}_{\mathcal{P}\sim\mathrm{Ber}(g_\theta(\mathcal{A},X,v))}[\cdot]$ by computing the expectation $\mathbb{E}_{\mathcal{U}\sim\mathrm{Unif}(0,1)}[\cdot]$ in (11). Following the routine of the Gumbel softmax trick [27], [35], $\mathbb{E}_{\mathcal{U}\sim\mathrm{Unif}(0,1)}[\cdot]$ can be estimated by first sampling the entries of $\mathcal{U}$ from the uniform distribution $\mathrm{Unif}(0,1)$ and then computing the loss term $L(f(Q(\mathcal{A},\mathcal{S}),X,v),y) + \lambda\max(\sum_{i=1}^{l}\|S_{R_i}\|_1 - \xi, 0)$. The time cost of this estimation is $(k+1)(|\mathcal{V}|+|\mathcal{E}|-1)+w \sim O(k|\mathcal{V}|+w)$ because it takes $k(|\mathcal{V}|+|\mathcal{E}|-1)$ time to compute Φ, $|\mathcal{V}|+|\mathcal{E}|-1$ time to compute the entries in $\mathcal{S}$ by (10), w time to compute the loss term and we have $|\mathcal{E}| \sim O(|\mathcal{V}|)$ due to the high sparsity of real-world graphs.

In summary, the time complexity of using the proposed method to estimate the expectation $\mathbb{E}_{\mathcal{P}\sim\mathrm{Ber}(g_\theta(\mathcal{A},X,v))}[\cdot]$ is $O(k|\mathcal{V}|+w)$, which grows linearly with $|\mathcal{V}|$. This is much more efficient than explicitly computing $\mathbb{E}_{\mathcal{P}\sim\mathrm{Ber}(g_\theta(\mathcal{A},X,v))}[\cdot]$. The optimization problem in (11) can be directly solved by gradient descent, thus enabling us to train the generator g_θ in an end-to-end manner.

E. Training Procedure

Now we present the procedure for training the generator g_θ, which is summarized in Algorithm 1.

Overall, we train g_θ by stochastic gradient descent. We first initialize the parameters θ of g_θ by the Kaiming initialization [36] (step 1), and set the number of training iterations t (step 2), the penalty parameter λ (step 3) and the temperature parameter τ (step 4) to one. During each training iteration, we first sample a batch of training nodes $\mathcal{V}_B$ (step 6). Then, for each target node $v \in \mathcal{V}_B$, we select the candidate edges of v according to the proposed two strategies (step 8). Next, we compute the average loss on $\mathcal{V}_B$ by (11) (step 10) and compute the gradients of θ by backpropagation (step 11). Finally, we use the ADAM optimizer [37] to update θ (step 12). For every 1,000 training iterations, we update the values of τ and λ (steps 14-15). The temperature parameter τ is gradually reduced to avoid introducing a large variance in the gradients computed early in the training and to make the smooth perturbation $\mathcal{S}$ closer to discrete as the training proceeds [27], [35]. We adopt the following annealing schedule for τ:

$$\tau(t) = \max\left(0.01, e^{-0.001t}\right), \tag{12}$$

where $\tau(t)$ is the value of τ at the t-th training iteration. The penalty parameter λ is increased by ten times. We update the number of training iterations t at the end of each training iteration (step 17).

Algorithm 1: Training the perturbation generator g_θ

Inputs : The victim HGNN f or the surrogate HGNN f_S, the heterogeneous graph $\mathcal{G}$ with the training nodes, the parameters κ, ϵ and β, and the budget ξ.
Output: The trained generator g_θ.

1 Initialize θ by the Kaiming initialization [36].
2 Initialize the number of training iterations $t = 1$.
3 Initialize the penalty parameter $\lambda = 1$.
4 Initialize the temperature parameter $\tau = 1$.
5 **do**
6 $\mathcal{V}_B \leftarrow$ Sample a batch of training nodes.
7 **for** *each target node* $v \in \mathcal{V}_B$ **do**
8 Select the candidate edges of v according to the proposed two strategies.
9 **end**
10 Compute the average loss on $\mathcal{V}_B$ by (11).
11 Compute the gradients with respect to θ.
12 Update θ by the ADAM optimizer [37].
13 **if** $t \bmod 1000 = 0$ **then**
14 Update τ by (12).
15 Update $\lambda \leftarrow 10 \times \lambda$.
16 **end**
17 Update $t \leftarrow t + 1$.
18 **while** *not converge*;
19 **return** $g_{\boldsymbol{\theta}}$.

F. Conducting Attacks

During the inference phase, we use the generator g_θ trained by Algorithm 1 to conduct attacks on victim HGNNs. For a new target node v_{new}, we can first compute $\Phi = g_\theta(\mathcal{A}, X, v_{\text{new}})$ based on the candidate edges selected by the proposed two strategies, and then sample a perturbation $\mathcal{P}$ according to the distribution $\mathrm{Ber}(\Phi)$ to perturb the heterogeneous graph $\mathcal{G}$. However, this raises two issues. First, $\mathcal{P}$ may add new edges that are not semantically meaningful. For example, in an academic network, $\mathcal{P}$ may add an edge between a paper

Algorithm 2: Conducting an attack for a node v_{new}

Inputs : The generator g_θ, the link predictor f_M, the heterogeneous graph $\mathcal{G}$, the new target node v_{new}, the parameters ϵ and β, the threshold δ and the budget ξ.

Output: A perturbed heterogeneous graph $\mathcal{G}'$.

1 Select the candidate edges of v_{new} according to the proposed two strategies.
2 Use g_θ to compute the probabilities in Φ corresponding to the candidate edges.
3 Sort the candidate edges by their corresponding probability values in Φ from largest to smallest.
4 Initialize the counter $c = 0$.
5 **for** *each candidate edge* e *in the sorted set* **do**
6 **if** e *exists* **then**
7 Delete e from the heterogeneous graph $\mathcal{G}$.
8 Update $c \leftarrow c + 1$.
9 **else if** e *does not exist* **then**
10 **if** $f_M(e) \geq \delta$ **then**
11 Add e to the heterogeneous graph $\mathcal{G}$.
12 Update $c \leftarrow c + 1$.
13 **end**
14 **if** $c = \xi$ **then**
15 break.
16 **end**
17 **end**
18 **return** $\mathcal{G}'$.

on medieval literature and a machine learning conference. However, since the medieval literature paper is unlikely to be published at the machine learning conference, the added edge is not semantically meaningful. Such semantic meaninglessness could make it easy to detect these malicious edges. We do not consider the semantic meaningfulness of deleting an edge because most real-world graphs are incomplete and missing information. Second, $\mathcal{P}$ may not adhere to the budget constraint by containing more than ξ edge modifications.

To tackle the first issue, we introduce an HGNN-based link predictor, denoted by f_M, to determine the semantic meaningfulness of adding a new edge e. Specifically, f_M is trained to perform link prediction task on the heterogeneous graph $\mathcal{G}$. The prediction score of f_M on e, denoted as $f_M(e) \in (0, 1)$, measures the semantic meaningfulness of adding e to $\mathcal{G}$. We consider adding e to be semantically meaningful if $f_M(e) \geq \delta$ and meaningless otherwise, where $0 < \delta < 1$ is a threshold.

The intuition behind this is that a well-trained link predictor captures the underlying connectivity patterns of the graph. Therefore, if two nodes share meaningful attributes and are likely to be connected based on graph topology, then the link predictor will score the edge between the two nodes highly [38], [39]. Accordingly, if f_M assigns a high prediction score to e, then e is likely to be a plausible edge within the given graph, ensuring that the modification of adding e maintains realism and is semantically meaningful.

To address the second issue, instead of sampling a perturbation $\mathcal{P}$ from the distribution $\text{Ber}(\Phi)$, we adopt the following steps to perturb exactly ξ candidate edges. Specifically, we first sort the candidate edges by their corresponding probability values in Φ from largest to smallest. Then, starting from the first candidate edge e, we perturb it if (1) e exists, or (2) e does not exist and $f_M(e) \geq \delta$. We repeat this process until a total number of ξ candidate edges are perturbed. In this way, only ξ edge modifications are performed, thus satisfying the budget constraint.

Algorithm 2 summarizes the process of launching an attack for a new target node v_{new}. We first select the candidate edges of v_{new} by the proposed strategies (step 1). Then, we compute the probabilities in Φ corresponding to the selected candidate edges (step 2). Next, we iteratively perturb ξ candidate edges following the steps presented above (steps 3-17).

G. Time Complexity Analysis

In this subsection, we analyze the time complexity of the proposed algorithms.

Time complexity of Algorithm 1. Denote by I the total number of training iterations, the time cost of Algorithm 1 mainly consists of the time costs of the I training iterations. In each training iteration, denoted by B the batch size, we use a batch of B target nodes $\mathcal{V}_B$ to train the generator g_θ. This process consists of four phases and we analyze the time cost of each of these phases as follows.

The first phase corresponds to steps 7-9 of Algorithm 1, where we use the proposed two strategies to select the candidate edges for each target node v in $\mathcal{V}_B$. For Strategy 1, we perform the breadth-first search (BFS) algorithm to find the existing edges in the ϵ-hop neighborhood of v. In the worst case where the subgraph induced by the ϵ-hop neighborhood is the entire graph, the time cost of the BFS is $|\mathcal{V}|+|\mathcal{E}|$ because it must traverse the entire graph. Thus, the worst-case time cost of performing Strategy 1 is $|\mathcal{V}| + |\mathcal{E}|$. For Strategy 2, since sampling a proportion β of non-existing edges connecting to v requires checking all nodes except v in the worst case, the worst-case time cost of performing Strategy 2 is $|\mathcal{V}| - 1$. Therefore, the worst-case time cost of the first phase to process B target nodes in $\mathcal{V}_B$ is $B(2|\mathcal{V}| + |\mathcal{E}| - 1)$.

The second phase corresponds to step 10 of Algorithm 1, where we compute the average loss on $\mathcal{V}_B$ by (11). This requires performing a forward pass for each target node in $\mathcal{V}_B$ by computing the expectation $\mathbb{E}_{\mathcal{U}\sim\text{Unif}(0,1)}[\cdot]$ in (11). As discussed in Section V-D, the worst-case time cost of estimating $\mathbb{E}_{\mathcal{U}\sim\text{Unif}(0,1)}[\cdot]$ is $(k+1)(|\mathcal{V}|+|\mathcal{E}|-1)+w$. Thus, the worst-case time cost of the second phase to forward pass B target nodes in $\mathcal{V}_B$ is $B((k+1)(|\mathcal{V}| + |\mathcal{E}| - 1) + w)$.

The third phase corresponds to step 11 of Algorithm 1, where we compute the gradients of the parameters θ for $\mathcal{V}_B$ via backpropagation. This typically has a time cost of roughly two times that of the forward passes in the second phase. Thus, the time cost of the third phase is $2B((k+1)(|\mathcal{V}|+|\mathcal{E}|-1)+w)$.

The fourth phase corresponds to step 12 of Algorithm 1, where we use the ADAM optimizer to update θ. The time cost of this update is proportional to the number of parameters of the generator g_θ, which is denoted as $|\theta|$. However, since the time cost of using g_θ to compute each probability in Φ is k,

which implicitly depends on $|\theta|$ because a larger g_θ increases k, we approximate the time cost of the update by k. Thus, the time cost of the fourth phase is k.

To summarize, the time cost of each training iteration is the sum of the time costs of the four phases described above, and the time cost of Algorithm 1 is the sum of the time costs of the I training iterations. Thus, the time complexity of Algorithm 1 is $I(B((3k+5)|\mathcal{V}|+(3k+4)|\mathcal{E}|+3(w-k)-4)+k) \sim O\big(IB(k|\mathcal{V}|+k|\mathcal{E}|+w)\big)$. Due to the high sparsity of real-world graphs, we have $|\mathcal{E}| \sim O\big(|\mathcal{V}|\big)$. Thus, the time complexity becomes $O\big(IB(k|\mathcal{V}|+w)\big)$. Since I and B are constants and k and w are not affected much by the size of the graph, the time complexity of Algorithm 1 grows linearly with $|\mathcal{V}|$, which makes Algorithm 1 time-efficient.

Time complexity of Algorithm 2. The time cost of Algorithm 2 mainly consists of the time costs of four phases, which are analyzed as follows.

The first phase corresponds to step 1 of Algorithm 2, where we use the proposed two strategies to select the candidate edges of the new target node v_{new}. As discussed earlier, the worst-case time cost of this step is $2|\mathcal{V}|+|\mathcal{E}|-1$.

The second phase corresponds to step 2 of Algorithm 2, where we use g_θ to compute the probabilities in Φ corresponding to the selected candidate edges. Since the maximum number of the candidate edges is equal to the maximum size of the reduced perturbation search space $|\mathcal{V}|+|\mathcal{E}|-1$, the worst-case time cost of the second phase is $k(|\mathcal{V}|+|\mathcal{E}|-1)$.

The third phase corresponds to step 3 of Algorithm 2, where we sort the candidate edges by their corresponding probabilities in Φ. Since the time cost of a sorting algorithm for n elements is typically $cn\log n$ where c is a constant depending on implementation details of the sorting algorithm, the time cost of the third phase is $c(|\mathcal{V}|+|\mathcal{E}|-1)\log(|\mathcal{V}|+|\mathcal{E}|-1)$.

The fourth phase corresponds to steps 5-16 of Algorithm 2, where we iteratively perturb ξ candidate edges based on the existences of the edges and the prediction scores of the link predictor f_M on the edges. Denoted by μ the maximum time cost of using f_M to score an edge. In the worst case, we need to iterate over all the candidate edges and use f_M to score each of them. Thus, the worst-case time cost of the fourth phase is $\mu(|\mathcal{V}|+|\mathcal{E}|-1)$.

To summarize, the time cost of Algorithm 2 is the sum of the time costs of the four phases described above. Thus, the time complexity of Algorithm 2 is $|\mathcal{V}|+(k+\mu+c\log(|\mathcal{V}|+|\mathcal{E}|-1)+1)(|\mathcal{V}|+|\mathcal{E}|-1) \sim O\big((k+\mu+c\log(|\mathcal{V}|+|\mathcal{E}|))(|\mathcal{V}|+|\mathcal{E}|)\big)$. Since $|\mathcal{E}| \sim O\big(|\mathcal{V}|\big)$, the time complexity becomes $O\big((k+\mu+c\log(|\mathcal{V}|))|\mathcal{V}|\big)$. Since c is a constant and k and μ are not affected much by the size of the graph, the time complexity of Algorithm 2 grows quasilinearly with $|\mathcal{V}|$, which makes Algorithm 2 time-efficient.

VI. Experiments

In this section, we evaluate the performance of the proposed GHAttack and compare it with other attack methods on multiple HGNNs and datasets. In particular, we wish to answer the following research questions: **Q1:** How effective is GHAttack in the white-box attack setting? **Q2:** How effective is GHAttack in the gray-box attack setting? **Q3:** How effective is GHAttack against different defenses? **Q4:** How fast is GHAttack in launching attacks? **Q5:** What is the distribution of edges perturbed by GHAttack? **Q6:** How does GHAttack perform with different hyperparameters? **Q7:** How scalable is GHAttack? **Q8:** How efficient is the training of GHAttack? **Q9:** How does GHAttack perform with different generator g_θ? **Q10:** How does the link predictor f_M affect GHAttack? Due to the page limit, we answer Q5-Q10 in Appendices A-F of the *supplementary material*.

TABLE I
Characteristics of the victim HGNNs in different dimensions of model design.

Victim HGNN	Meta-path usage	Attention mechanism	Model Architecture	Model Complexity
RGCN	None	None	GCN	Simple
HGT	None	Relation-level	Transformer	Complex
SimpleHGN	None	Relation-level	GCN	Simple
HAN	Explicit	Meta-path-level	GCN	Simple
HAN-RoHe	Explicit	Meta-path-level	GCN	Simple
GTN	Implicit	Relation-level	Transformer	Complex
HetSANN	Implicit	Type-aware	GCN	Complex
MHNF	Implicit	Hierarchical	GCN	Complex
AGAT	None	Aspect-aware	GCN	Simple
HAGNN	Explicit	Hierarchical	GCN	Complex

A. Experimental Settings

Victim HGNNs. We use ten representative HGNNs for evaluation, including RGCN [32], HGT [8], SimpleHGN [31], HAN [1], HAN-RoHe [12], GTN [30], HetSANN [2], MHNF [7], AGAT [5] and HAGNN [6]. Among them, HAN-RoHe is a robust model that is specifically designed to resist structure-based adversarial attacks. We choose these HGNNs as victim models because they exhibit good diversity across several dimensions of design, as shown in Table I. The hyperparameters of each victim HGNN are adopted from its original implementation.

Datasets. We adopt six heterogeneous graph datasets that are used for node classification. As shown in Table II, these datasets are diversified in terms of graph statistics, class distribution of target nodes, presence of noise and semantic domains. We elaborate their detailed descriptions as follows.

- ACM [40]. This is a citation network consisting of nodes of types "Paper", "Author" and "Subject" and edges reflecting "Paper-Author" and "Paper-Subject" relations. The classification task is to predict the venue of each paper. We use the original version of ACM provided in [40].
- IMDB [40]. This is a movie network consisting of nodes of types "Movie", "Actor" and "Director" and edges reflecting "Movie-Actor" and "Movie-Director" relations. The classification task is to predict the genre of each movie. We use the original version of IMDB provided in [40].
- DBLP [1]. This is also a publication network consisting of nodes of types "Paper", "Author", "Conference" and "Term" and edges reflecting "Paper-Author", "Paper-Conference" and "Paper-Term" relations. We use it to conduct two classification tasks. The first task is to predict the research

TABLE II
CHARACTERISTICS OF THE HETEROGENEOUS GRAPH DATASETS. THE NODE TYPE ENTROPY IS THE ENTROPY OF THE DISTRIBUTION OF NODE TYPES. THE RELATION ENTROPY IS THE ENTROPY OF THE DISTRIBUTION OF RELATIONS. THE CLASS ENTROPY IS THE ENTROPY OF THE CLASS DISTRIBUTION OF TARGET NODES.

Dataset	# Nodes	# Edges	# Node types	Node type entropy	# Node attributes	# Relations	Relation entropy	Average node degree	# Classes	Class entropy	Noised	Domain
ACM	8,994	12,961	3	0.97	1,902	2	0.78	1.44	3	1.58	No	Citation network
IMDB	12,772	18,644	3	1.49	1,256	2	0.81	1.45	3	1.47	No	Movie network
DBLP-Paper	26,128	119,783	4	1.42	334	3	1.14	4.58	4	0.85	No	Publication network
DBLP-Author	26,128	119,783	4	1.42	334	3	1.14	4.58	4	0.69	No	Publication network
Freebase	12,164,758	31,491,283	8	2.19	n/a	18	2.63	2.59	8	0.16	No	General knowledge
Freebase-Noise	12,164,758	31,491,283	8	2.19	n/a	18	2.63	2.59	8	0.16	Yes	General knowledge

area of each author, where the label of each author is provided in [1]. The second task is to predict the research area of each paper, where the label of each paper is obtained by classifying the title of the paper into four research areas. We refer to the datasets corresponding to the first and second tasks as *DBLP-Author* and *DBLP-Paper*, respectively.

- Freebase [40]. This is a large-scale knowledge graph extracted from the Internet. It consists of nodes of 8 types and edges reflect 18 relations. The classification task is to predict the genre of each node of type "Book". In addition to using the original version of Freebase provided in [40], we construct a noisy version of it as follows. For the bipartite graph corresponding to each relation of Freebase, we first randomly delete 10% of existing edges and then randomly add the same number of non-existing edges to the bipartite graph. We refer to the noisy version of Freebase as *Freebase-Noise*.

Each dataset is split into a training set, a validation set and a testing set. The training set is used to train victim HGNNs and parametric models of model-based attack methods. The validation set is used for hyperparameter tuning. The testing set is used to evaluate the performance of different attack methods. For DBLP-Paper, we use 800/400/13,128 nodes for training/validation/testing; for Freebase and Freebase-Noise, we use 1,200/400/8,967 nodes for training/validation/testing. For the other datasets, we use their default splitting ratios [1], [40].

Each dataset has a set of meta-paths, which are required for meta-path-based HGNNs such as HAN [1]. For DBLP-Paper, we adopt the meta-paths {APA, APCPA, APTAP}, where A, P, C, T denote "Author", "Paper", "Conference" and "Term", respectively. For the other datasets, we adopt the meta-paths from their original implementations [1], [40].

Baselines. We compare the proposed GHAttack with both optimization-based methods including FGSM [12], HGAttack [13] and HSA [14], and model-based methods including GUA [20], CD-ATTACK [23], RL-S2V [24] and PR [21]. These model-based methods are designed to attack classic graph neural networks on homogeneous graphs and thus cannot handle graph heterogeneity. Hence, we apply them to attack HGNNs by training one model on each relation of a heterogeneous graph, such that each model focuses on perturbing edges on the corresponding relation. To report the final performance on the testing set, we use the best model that achieves the highest attack effectiveness on the validation set.

Our methods. In addition to the proposed GHAttack, we develop a variant of it that cannot leverage graph heterogeneity. Specifically, similar to the model-based baseline methods, we train one generator on each relation of a heterogeneous graph, perturbing only the edges on that relation, and report the final performance on the testing set using the generator that achieves the highest attack effectiveness on the validation set. We refer to this degenerated version of GHAttack as *GHAttack-D*.

Evaluation metrics. We measure the attack effectiveness of a method against a victim HGNN by the **Micro-F1** [12] of the victim HGNN achieved on the testing set. The value of Micro-F1 ranges from 0 to 1, where a smaller value indicates worse predictive performance of the victim HGNN and thus implies better attack effectiveness of that method. We report Micro-F1 in percentage by default. We measure the attack efficiency of a method by **attack time (AT)**, which is the total time cost of generating the perturbed heterogeneous graphs for all target nodes in the testing set. A lower AT implies better attack efficiency. We report AT in minutes by default.

Implementation details. Here, we introduce the implementation details of the baseline methods and our methods.

For the compared optimization-based methods, we use the source code of FGSM and we implement HGAttack and HSA to achieve performance comparable to that reported in their respective papers. For the compared model-based methods, we use the source code of GUA, CD-ATTACK and RL-S2V, and we implement PR to achieve performance comparable to that reported in [21]. Since these model-based methods use different settings, we unify them as follows for the fairness of comparison. For GUA, we choose a number of ξ anchor nodes that correspond to the top-ξ most effective nodes learned by GUA, in order to use the same budget ξ as the other methods. For CD-ATTACK, we replace the hiding loss in [23] by the loss function defined in (8), so that it can launch attacks for the node classification task. For RL-S2V, we adopt the PBA-C attack setting in [24], as the class probabilities predicted by the victim HGNNs are accessible in our experimental settings. For PR, we repurpose it to attack the victim HGNNs in the node classification task following the same way as in [24]. In addition, HGAttack, HSA and CD-ATTACK only assume the gray-box setting and conduct transfer attacks. In the white-box setting, since the victim HGNNs are accessible, we use HGAttack, HSA and CD-ATTACK to attack them directly instead of attacking a surrogate HGNN.

For our methods, we adopt the backbone of HGT [8] as the HGNN backbone h_η of the generator g_θ by default and set

TABLE III
MICRO-F1 (%) OF ALL THE METHODS WHEN THE BUDGET $\xi = 5$. A LOWER MICRO-F1 MEANS HIGHER ATTACK EFFECTIVENESS. BOLD NUMBERS INDICATE THE LOWEST MICRO-F1 WHEN ATTACKING EACH VICTIM HGNN ON EACH DATASET.

Dataset	Attack method	RGCN	HGT	SimpleHGN	HAN	HAN-RoHe	GTN	HetSANN	MHNF	AGAT	HAGNN	Mean Micro-F1
ACM	No attack	92.05	90.54	91.72	89.98	88.53	91.11	92.00	90.40	90.24	91.64	90.82
	FGSM	41.53	73.50	55.25	51.56	87.20	74.44	39.78	39.82	33.68	39.32	53.60
	HGAttack	33.96	53.72	**9.68**	**17.42**	83.36	65.73	**24.72**	19.26	27.34	35.07	37.03
	HSA	32.42	55.29	12.05	20.13	84.93	61.20	26.17	**18.23**	26.39	37.44	37.37
	GUA	40.42	65.05	34.69	33.41	85.98	65.22	41.29	37.40	36.29	55.28	49.50
	CD-ATTACK	35.92	58.37	23.81	25.75	83.64	62.32	36.38	33.60	30.45	42.42	43.26
	RL-S2V	43.29	72.38	43.79	47.22	87.27	64.50	42.76	41.60	43.58	68.20	55.46
	PR	33.45	55.83	26.34	28.06	83.53	59.82	37.28	36.05	27.52	40.44	42.83
	GHAttack-D (Our)	36.20	56.84	21.56	22.10	84.73	62.92	32.20	28.19	32.30	39.32	41.64
	GHAttack (Our)	**31.90**	**50.32**	17.72	18.10	**82.62**	**57.24**	25.78	20.12	**23.40**	**33.50**	**36.07**
IMDB	No attack	60.09	61.39	61.99	59.77	53.64	60.20	59.94	61.22	59.40	61.26	59.89
	FGSM	14.35	19.50	33.37	24.75	41.90	31.83	17.53	51.90	26.06	31.83	29.30
	HGAttack	11.86	9.80	**7.03**	**8.42**	37.58	17.95	11.48	18.66	13.86	**17.51**	15.42
	HSA	13.25	10.82	8.93	11.31	35.29	19.01	12.45	**17.23**	15.57	18.04	16.19
	GUA	16.09	19.97	25.80	14.55	39.29	26.06	18.77	40.70	30.58	33.05	26.49
	CD-ATTACK	15.47	9.42	14.98	11.43	36.06	25.27	17.29	42.66	23.70	27.24	22.35
	RL-S2V	20.75	25.10	31.68	23.13	42.30	33.22	19.41	53.77	37.44	39.72	32.65
	PR	16.23	14.57	17.29	13.04	35.90	25.83	16.15	44.30	22.58	25.73	23.16
	GHAttack-D (Our)	15.20	10.52	14.50	12.79	35.82	24.42	15.20	39.30	20.38	28.52	21.67
	GHAttack (Our)	**11.84**	**7.02**	8.46	9.69	**33.75**	**15.44**	**10.64**	21.05	**13.26**	18.40	**14.96**
DBLP-Paper	No attack	74.32	74.41	75.69	71.53	70.41	73.90	73.86	74.14	72.02	73.58	73.37
	FGSM	17.30	36.90	33.62	18.25	69.02	30.50	29.38	37.22	19.33	25.40	31.69
	HGAttack	7.82	17.60	13.49	8.75	**62.28**	17.47	15.30	17.72	9.47	12.45	18.24
	HSA	**6.56**	14.86	14.24	8.92	64.05	19.97	14.11	19.05	11.22	11.92	18.49
	GUA	20.19	30.35	28.05	19.20	68.04	30.42	34.93	32.80	23.47	30.82	31.83
	CD-ATTACK	16.58	23.84	16.50	13.85	67.29	25.06	21.36	27.30	17.72	20.42	24.99
	RL-S2V	23.47	30.52	32.68	21.64	68.93	31.40	38.37	35.39	28.20	34.90	34.55
	PR	14.73	22.26	18.90	16.04	66.73	24.28	35.93	24.30	20.14	24.51	26.78
	GHAttack-D (Our)	14.92	22.04	15.85	10.24	68.80	25.50	19.80	23.25	20.39	18.79	23.96
	GHAttack (Our)	6.85	**9.46**	**11.80**	**6.32**	64.95	**15.16**	**11.05**	**13.47**	**7.22**	**11.68**	**15.80**
DBLP-Author	No attack	90.45	91.18	93.84	89.78	90.58	90.80	92.27	91.71	88.56	90.34	90.95
	FGSM	29.90	39.20	38.57	20.52	88.29	41.58	36.32	30.52	17.92	31.94	37.48
	HGAttack	**9.24**	22.80	**16.22**	**8.44**	82.78	23.30	19.94	21.05	14.18	**20.61**	23.86
	HSA	11.72	18.55	16.89	11.25	81.29	21.80	21.45	20.64	15.39	21.20	24.02
	GUA	13.23	20.07	32.54	21.93	85.08	30.29	23.10	21.36	19.27	28.37	29.52
	CD-ATTACK	12.20	20.85	23.81	15.05	81.36	16.18	18.05	19.97	14.26	23.53	24.53
	RL-S2V	26.27	34.75	40.68	24.61	87.88	38.39	39.41	28.32	23.85	35.60	37.98
	PR	13.60	18.58	26.09	17.73	82.37	21.79	20.94	18.60	15.07	24.90	25.97
	GHAttack-D (Our)	13.69	17.05	22.10	13.19	81.16	15.35	16.03	16.73	13.60	23.55	23.25
	GHAttack (Our)	12.20	**16.94**	21.62	12.82	**80.35**	**15.20**	**14.07**	**15.45**	**12.67**	21.38	**22.27**
Freebase	No attack	69.04	73.53	71.49	68.90	67.25	72.95	69.20	70.20	68.22	70.52	70.13
	FGSM	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	HGAttack	22.40	24.05	**16.19**	**24.70**	57.82	27.50	22.03	25.75	23.69	24.22	26.84
	HSA	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	GUA	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	CD-ATTACK	45.30	43.69	42.06	47.24	58.63	44.30	46.38	48.57	43.58	51.20	47.10
	RL-S2V	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	PR	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	GHAttack-D (Our)	27.44	30.08	27.30	32.64	61.75	32.94	26.33	33.43	31.24	30.73	33.99
	GHAttack (Our)	**19.29**	**23.40**	17.37	25.44	**52.05**	**22.29**	**18.57**	**24.72**	**20.21**	**22.83**	**24.62**
Freebase-Noise	No attack	75.26	73.60	73.53	74.57	75.63	74.29	71.52	73.50	74.25	72.36	73.88
	FGSM	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	HGAttack	21.37	22.28	**15.62**	23.90	54.37	25.66	21.24	22.73	20.43	22.70	25.03
	HSA	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	GUA	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	CD-ATTACK	42.28	39.19	40.38	44.20	55.23	42.05	43.74	45.60	43.04	48.25	44.40
	RL-S2V	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	PR	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	GHAttack-D (Our)	26.28	28.35	24.87	29.41	59.20	31.04	23.76	31.82	29.20	29.77	31.37
	GHAttack (Our)	**17.55**	**22.04**	16.50	**23.74**	**50.82**	**21.38**	**16.33**	**21.29**	**18.85**	**21.63**	**23.01**

the embedding dimension H in (4) to 256. We adopt another HGT as the link predictor f_M used in Algorithm 2. On each dataset, f_M is trained to perform link prediction task following the implementation in [31] and we set the threshold $\delta = 0.8$ by default. We use 400 training epochs for IMDB and 200 for the other datasets. For the hyperparameters ϵ and β used in the proposed two strategies, we set $\epsilon = \text{dia}(\mathcal{G})$ and $\beta = 1$ on ACM, IMDB, DBLP-Paper and DBLP-Author, where dia($\mathcal{G}$)

denotes the diameter of the heterogeneous graph $\mathcal{G}$; and we set $\epsilon = 2$ and $\beta = 0.1$ on Freebase and Freebase-Noise. In all experiments, we set the κ of the loss function in (8) to 5, use a batch size of 32 and set the learning rate of the ADAM optimizer to 10^{-3}. For all the compared methods, we use the same budget $\xi = 5$ if not otherwise specified. GHAttack is implemented using Pytorch version 1.12.1, DGL version 0.9.1 with CUDA version 11.3. All experiments are conducted on a server with an NVIDIA RTX 3090 GPU, 64GB main memory and an Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz.

B. Attack Effectiveness in the White-box Setting (Q1)

Table III compares the attack effectiveness of our methods with the baseline methods in the white-box setting, where the results of the optimization-based methods and the model-based methods are grouped separately. As reference, we also show the Micro-F1 when the victim HGNNs are not under attack, and report the mean Micro-F1 of each method in attacking all the victim HGNNs. On Freebase and Freebase-Noise, we cannot report the results of FGSM, HSA, GUA, RL-S2V and PR because they require too much memory to run on these large-scale graphs, which causes out-of-memory issues. From the results, we have the following observations.

GHAttack outperforms the compared model-based methods by a large margin. Compared with the second-best model-based method GHAttack-D, GHAttack achieves an average improvement of 22.8% in achieving lower Micro-F1. This success can be mainly attributed to the ability of GHAttack to perturb edges on heterogeneous relations, whereas the other model-based methods can only perturb edges on a single one. However, such advantage is relatively less significant on DBLP-Author. This is because, on DBLP-Author, there is only one relation "Paper-Author" associated with the nodes of type "Author" that need to be classified. Hence, perturbing edges on the other relations offers limited improvement in attack effectiveness, thus restraining the potential of GHAttack to exploit graph heterogeneity. We will further discuss this phenomenon in Appendix A.

The performance differences between the other model-based methods exhibit a similar trend across different datasets and victim HGNNs. CD-ATTACK, PR and GHAttack-D achieve comparable results in most cases, as they all train a model on a single relation to produce perturbations. However, on Freebase and Freebase-Noise, GHAttack-D performs much better than CD-ATTACK. This is because, in order to run on large-scale graphs, CD-ATTACK is restricted to only deleting edges [23], whereas GHAttack-D allows both edge addition and deletion. This offers greater potential for GHAttack-D to find more effective edges to perturb. GUA has inferior performance to CD-ATTACK, PR and GHAttack-D because its attack scheme, which alters connections between target nodes and a fixed set of anchor nodes, is limited in attack flexibility and thus reduces its effectiveness. RL-S2V performs the worst among all the model-based methods, as it employs a reinforcement learning approach that does not effectively leverage the gradient information of the victim HGNNs.

GHAttack achieves comparable performance to the best optimization-based method. As shown in Table III, in most cases, GHAttack achieves close or even lower Micro-F1 when compared with the best optimization-based method HGAttack. Moreover, as shown by the reported mean Micro-F1, GHAttack consistently outperforms FGSM, HGAttack and HSA. This indicates the good generalization ability of the generator g_θ used in GHAttack, which can produce effective perturbations for new target nodes. In comparison, FGSM is inferior to HGAttack and HSA, as it uses a simple approach to optimize perturbations in pursuit of high speed rather than high effectiveness.

C. Attack Effectiveness in the Gray-box Setting (Q2)

To evaluate the attack effectiveness of different methods in the gray-box setting, we conduct transfer attacks on the victim HGNNs. Specifically, for each optimization-based method, it first optimizes perturbations to attack a surrogate HGNN f_S, and then uses the optimized perturbations to attack a victim HGNN f. For each model-based method, it first trains a model to attack the surrogate HGNN f_S and then uses the trained model to generate perturbations to attack the victim HGNN f. Denoted by Micro-F1$_{f_S,f}$ the Micro-F1 of f achieved on the testing set by conducting the above transfer attacks and by $\mathcal{F}$ the set of ten victim HGNNs used in our experiments, we measure the transfer attack effectiveness of a method when using a surrogate HGNN $f_S \in \mathcal{F}$ by

$$\text{Trans}_{f_S} = \frac{1}{9} \sum_{f \in \mathcal{F} \setminus f_S} \text{Micro-F1}_{f_S,f}, \tag{13}$$

which is the average of Micro-F1$_{f_S,f}$ when using the perturbations generated for f_S to attack each of the other nine victim HGNNs. A smaller Trans$_{f_S}$ means better transfer attack effectiveness. In particular, HGAttack and HSA specifically design a surrogate model for launching transfer attacks, rather than adopting an existing HGNN [13], [14]. Hence, we conduct another set of experiments for HGAttack and HSA, where we consistently use their specially designed surrogate model instead of an HGNN in $\mathcal{F}$ as f_S, but we still measure the transfer attack effectiveness by (13). We refer to these methods as HGAttack-S and HSA-S, respectively. Table IV compares the transfer attack effectiveness of different methods. Due to the out-of-memory issues, we cannot report the results of FGSM, HSA, HSA-S, GUA, RL-S2V and PR on Freebase and Freebase-Noise. We have the following findings.

GHAttack achieves better attack transferability than the other model-based methods. Compared with the second-best model-based method GHAttack-D, GHAttack obtains an average improvement of 21.4% in achieving lower Trans$_{f_S}$. This suggests that in the transfer attacks, edges perturbed on different relations are more effective than edges perturbed on a single relation. We believe that this is because edges on a relation that have a large effect on the surrogate HGNN f_S may have a small effect on the victim HGNN f, as f may rely primarily on messages passed on the other relations to make predictions. Hence, the ability of GHAttack to perturb edges on heterogeneous relations grants it superior transfer attack effectiveness. In addition, we observe a similar phenomenon as in Table III, where the advantage of GHAttack is relatively

TABLE IV

TRANS$_{f_S}$ (%) OF ALL THE METHODS WHEN THE BUDGET $\xi = 5$. A LOWER TRANS$_{f_S}$ MEANS HIGHER TRANSFER ATTACK EFFECTIVENESS. BOLD NUMBERS INDICATE THE LOWEST TRANS$_{f_S}$ WHEN USING EACH HGNN AS THE SURROGATE HGNN TO ATTACK THE OTHER HGNNS.

Dataset	Attack method	RGCN	HGT	SimpleHGN	HAN	HAN-RoHe	GTN	HetSANN	MHNF	AGAT	HAGNN	Mean Trans$_{f_S}$
ACM	FGSM	71.26	64.30	69.25	70.84	78.68	63.75	68.60	70.05	64.29	65.07	68.56
	HGAttack	52.94	47.51	**41.25**	49.78	73.11	50.30	47.40	48.36	50.66	51.86	51.32
	HGAttack-S	48.50	**43.62**	44.04	46.08	70.30	44.85	50.14	**43.94**	49.82	**48.33**	48.96
	HSA	54.62	49.03	42.89	47.20	71.19	48.52	50.22	49.76	53.59	53.20	52.02
	HSA-S	53.20	44.79	46.37	46.79	72.64	43.20	51.96	47.64	52.85	51.36	52.78
	GUA	64.53	59.90	59.70	63.95	81.15	60.38	72.51	64.47	62.44	67.36	65.64
	CD-ATTACK	60.96	54.60	57.80	62.50	77.05	63.77	68.42	55.05	57.50	62.74	62.04
	RL-S2V	68.32	65.49	70.48	74.51	80.23	72.58	66.46	64.56	69.29	71.11	70.30
	PR	62.55	53.80	55.26	63.15	78.54	61.30	70.38	59.72	58.37	64.91	62.80
	GHAttack-D (Our)	57.25	55.31	52.65	60.60	75.84	53.45	54.72	51.22	59.42	60.05	58.05
	GHAttack (Our)	**45.04**	48.45	43.64	**43.04**	**67.02**	**40.63**	**48.30**	46.32	**46.20**	51.47	**48.01**
IMDB	FGSM	36.61	32.40	39.75	41.82	40.08	33.95	34.31	36.48	34.75	33.40	36.36
	HGAttack	26.78	25.45	21.48	29.23	38.59	23.21	25.64	20.20	21.50	26.93	25.90
	HGAttack-S	21.40	24.30	**13.28**	**20.12**	35.58	25.36	22.04	23.85	22.76	25.88	23.46
	HSA	28.25	23.92	23.40	28.04	37.93	24.59	22.80	**19.85**	24.03	27.86	26.07
	HSA-S	26.28	23.33	23.90	23.74	39.05	23.97	25.28	21.83	19.86	27.12	25.43
	GUA	32.66	29.34	34.37	36.24	39.40	28.06	30.78	33.86	32.44	36.24	33.34
	CD-ATTACK	26.52	31.27	28.76	34.13	38.75	32.49	27.66	30.15	24.83	32.29	30.69
	RL-S2V	34.49	37.05	40.56	38.23	40.72	33.01	36.25	36.80	33.29	37.23	36.76
	PR	29.60	25.79	29.33	32.94	38.23	30.51	24.23	31.94	28.03	31.86	30.25
	GHAttack-D (Our)	25.33	32.74	25.05	35.20	38.75	30.85	24.35	29.13	26.07	33.68	30.12
	GHAttack (Our)	**18.94**	**16.04**	17.32	24.20	**33.80**	**21.40**	**19.92**	23.05	**17.22**	**24.29**	**21.62**
DBLP-Paper	FGSM	39.64	36.60	41.17	43.48	55.85	33.44	38.32	40.89	33.51	39.87	40.28
	HGAttack	27.78	20.93	23.87	26.13	50.33	26.97	23.84	22.43	22.30	24.90	26.95
	HGAttack-S	29.30	25.52	21.90	28.44	52.57	22.53	25.80	18.53	19.75	22.46	26.68
	HSA	30.94	21.15	25.21	**25.03**	53.69	25.04	28.80	25.37	20.11	23.72	27.90
	HSA-S	32.47	22.60	24.33	26.19	52.08	27.94	32.39	23.04	24.72	25.80	29.16
	GUA	36.76	34.53	39.48	37.78	53.50	40.80	35.13	33.76	36.29	38.22	38.63
	CD-ATTACK	27.78	28.60	33.17	36.60	54.90	34.05	38.55	35.34	31.58	29.68	35.03
	RL-S2V	42.23	37.38	35.20	40.62	55.33	37.92	34.45	40.90	38.20	41.69	40.39
	PR	31.66	32.08	33.62	41.75	56.20	38.01	35.90	39.22	34.29	35.83	37.86
	GHAttack-D (Our)	30.52	26.72	30.43	33.92	55.24	31.39	33.05	37.48	28.60	30.05	33.74
	GHAttack (Our)	**19.70**	**15.56**	**17.75**	25.08	**48.83**	**21.62**	**23.24**	**18.10**	**16.26**	**19.47**	**22.56**
DBLP-Author	FGSM	40.14	43.22	41.47	47.08	75.80	44.28	44.11	40.52	35.03	41.46	45.31
	HGAttack	34.04	26.52	33.38	32.68	68.35	26.39	35.48	33.63	31.11	34.68	35.63
	HGAttack-S	29.75	25.35	**29.23**	30.38	70.33	28.88	31.84	**25.40**	28.20	30.46	32.98
	HSA	37.79	27.20	32.59	33.80	70.47	29.80	34.65	32.91	34.22	35.20	36.86
	HSA-S	39.50	30.04	31.84	34.22	69.94	32.05	37.63	34.04	33.29	37.77	38.03
	GUA	34.77	33.02	37.20	39.40	74.45	32.80	36.23	40.68	33.67	35.03	39.73
	CD-ATTACK	29.11	24.94	31.44	33.03	75.28	35.12	**27.39**	32.60	26.25	31.20	34.64
	RL-S2V	43.67	45.92	43.51	49.25	76.45	45.90	40.57	48.36	41.49	45.90	48.10
	PR	31.66	27.04	32.69	32.15	73.20	38.85	28.39	34.27	28.12	32.62	35.90
	GHAttack-D (Our)	31.25	27.13	33.03	34.59	73.50	37.04	33.69	30.20	28.69	29.24	35.84
	GHAttack (Our)	**28.02**	**24.35**	31.68	**29.68**	**68.05**	**24.82**	31.24	29.95	**24.45**	**28.94**	**32.12**
Freebase	FGSM	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	HGAttack	36.28	35.40	36.16	39.66	61.93	37.50	43.04	38.63	39.30	37.36	40.53
	HGAttack-S	**33.73**	36.90	35.58	39.85	**59.28**	40.33	41.20	35.72	36.84	**35.24**	39.47
	HSA	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	HSA-S	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	GUA	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	CD-ATTACK	54.13	51.20	53.58	59.33	64.75	52.80	56.22	58.90	55.38	57.95	56.42
	RL-S2V	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	PR	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	GHAttack-D (Our)	45.73	39.30	48.25	42.73	63.04	43.90	48.26	44.06	44.57	48.39	46.82
	GHAttack (Our)	36.22	**33.80**	**32.94**	**38.06**	60.35	**36.44**	**37.32**	**33.26**	**35.20**	36.09	**37.97**
Freebase-Noise	FGSM	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	HGAttack	34.63	33.04	35.73	38.60	59.25	35.07	41.82	35.40	38.22	35.29	38.71
	HGAttack-S	**32.52**	33.29	34.20	**37.45**	58.93	39.20	38.37	33.70	34.90	**32.71**	38.75
	HSA	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	HSA-S	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	GUA	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	CD-ATTACK	52.72	48.50	51.82	58.55	63.30	48.96	53.04	53.73	54.40	55.82	54.08
	RL-S2V	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	PR	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	GHAttack-D (Our)	43.58	38.12	48.02	41.38	61.69	40.50	47.24	41.52	41.65	46.20	44.99
	GHAttack (Our)	33.84	**29.50**	**32.27**	38.44	**57.27**	**33.03**	**35.42**	**32.10**	**33.58**	33.23	**35.87**

less significant on DBLP-Author. We will discuss this in depth in Appendix A.

TABLE V
MEAN MICRO-F1 (%) OF ALL THE METHODS AGAINST DIFFERENT DEFENSES WHEN THE BUDGET $\xi = 5$. A LOWER MEAN MICRO-F1 MEANS HIGHER ATTACK EFFECTIVENESS AGAINST THE DEFENSE. BOLD NUMBERS INDICATE THE LOWEST MEAN MICRO-F1 AGAINST EACH DEFENSE ON EACH DATASET.

Dataset	Defense	FGSM	HGAttack	HSA	GUA	CD-ATTACK	RL-S2V	PR	GHAttack-D (Our)	GHAttack (Our)
ACM	No defense	53.60	37.03	37.37	49.50	43.26	55.46	42.83	41.64	**36.07**
	Jaccard	70.65	55.39	59.70	70.65	57.20	68.50	55.56	56.68	**54.22**
	First-order proximity	75.22	63.24	65.08	77.27	66.63	71.24	63.70	65.15	**61.40**
	Second-order proximity	78.74	64.58	64.96	78.14	71.24	74.30	68.51	69.40	**63.47**
IMDB	No defense	29.30	15.42	16.19	26.49	22.35	32.65	23.16	21.67	**14.96**
	Jaccard	37.52	30.23	31.77	39.70	36.56	39.68	37.60	35.25	**29.20**
	First-order proximity	43.27	37.05	40.32	44.85	42.28	43.27	41.94	40.12	**35.42**
	Second-order proximity	45.90	37.82	39.47	47.41	42.92	46.20	45.82	43.25	**36.93**
DBLP-Paper	No defense	31.69	18.24	18.49	31.83	24.99	34.55	26.78	23.96	**15.80**
	Jaccard	47.35	41.50	40.69	50.24	44.47	48.45	43.55	43.65	**39.25**
	First-order proximity	48.36	43.24	44.58	53.82	49.20	51.28	47.13	46.32	**42.83**
	Second-order proximity	51.94	45.93	48.31	56.20	50.73	52.70	49.84	48.13	**45.20**
DBLP-Author	No defense	37.48	23.86	24.02	29.52	24.53	37.98	25.97	23.25	**22.27**
	Jaccard	60.42	**46.52**	47.96	54.82	50.32	56.40	49.73	48.68	47.24
	First-order proximity	63.75	50.44	54.62	58.75	52.69	58.52	55.07	50.70	**48.30**
	Second-order proximity	64.94	53.05	54.70	59.06	55.04	59.66	56.41	52.25	**51.37**
Freebase	No defense	n/a	26.84	n/a	n/a	47.10	n/a	n/a	33.99	**24.62**
	Jaccard	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	First-order proximity	n/a	53.29	n/a	n/a	59.22	n/a	n/a	55.05	**49.50**
	Second-order proximity	n/a	54.45	n/a	n/a	60.47	n/a	n/a	57.82	**52.24**
Freebase-Noise	No defense	n/a	25.03	n/a	n/a	44.40	n/a	n/a	31.37	**23.01**
	Jaccard	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	First-order proximity	n/a	50.28	n/a	n/a	58.05	n/a	n/a	54.30	**44.68**
	Second-order proximity	n/a	52.47	n/a	n/a	61.54	n/a	n/a	58.16	**46.24**

GHAttack outperforms the optimization-based methods in the transfer attacks. As shown in Table IV, GHAttack achieves lower Trans_{f_S} than that of the optimization-based methods in most cases and consistently obtains the lowest mean Trans_{f_S} on different datasets. Such results indicate the better transferability of the perturbations produced by the generator g_θ of GHAttack. One possible reason is that since each perturbation generated by the optimization-based methods is specifically optimized to attack the surrogate HGNN f_S for a single target node, it may easily overfit to f_S and thus fail to attack other victim HGNNs. In contrast, the generator g_θ is trained to attack f_S for all the target nodes in the training set, which reduces the risk of overfitting each generated perturbation to f_S.

D. Attack Effectiveness Against Defense (Q3)

In this subsection, we investigate the attack effectiveness of the compared methods against different defenses. Since there are no defenses specially designed to resist evasion adversarial attack against HGNNs on heterogeneous graphs, we extend three defense methods [16], [41] on homogeneous graphs such that they can detect target nodes suspected to be attacked in heterogeneous graphs. The three defenses and the method to extend them are described as follows.

Jaccard [16] computes the similarity scores of node features between a target node v and its neighboring nodes, and considers v as attacked if the similarity score with one of the neighbors is below a threshold. First-order proximity [41] measures information discrepancy between v and its neighbors by computing the mean of the KL divergences between the classification probabilities of v and those of its neighbors; and Second-order proximity [41] computes the mean of the KL divergences between the classification probabilities of pairs of neighbors. Both of them consider v as attacked if the mean of the KL divergences is above a threshold. In a heterogeneous graph, since a target node v and its neighbors may be of different types, it is not always feasible to compute the similarity score in Jaccard and the KL divergence in First-order proximity and Second-order proximity. Hence, we extend these defense methods by redefining the neighbors of v as the set of nodes that are of the same type as v and have the shortest path from it. The thresholds used in the extended defense methods are tuned on the validation set to achieve the highest detection accuracy while having a false alarm rate below 10^{-3}.

Table V compares the attack effectiveness of different methods against the above defense methods, where we adopt the experimental settings in Section VI-B and report the mean Micro-F1 that is computed in the same way as in Table III. Due to the out-of-memory issues, we cannot report the results of FGSM, HSA, GUA, RL-S2V and PR on Freebase and Freebase-Noise. In addition, since Freebase and Freebase-Noise do not provide node features, we cannot report the results of Jaccard on these datasets. We can see that, compared with when no defenses are in place, the mean Micro-F1 of all the methods increases when a defense is applied, indicating that the extended defense methods are effective to some extent against adversarial attacks on HGNNs. Nevertheless, GHAttack still outperforms the other model-based methods and achieves comparable effectiveness to the best optimization-based method HGAttack. Moreover, the lowest mean Micro-F1 is achieved by GHAttack in most of the cases. Such superior performance of GHAttack is because, although its attack effectiveness is reduced by the defenses, the key mechanism of GHAttack that exploits graph heterogeneity to perturb edges

TABLE VI
MEAN AT (MINUTES) OF ALL THE METHODS IN THE WHITE-BOX AND THE GRAY-BOX SETTINGS WHEN THE BUDGET $\xi = 5$. A LOWER MEAN AT MEANS HIGHER ATTACK EFFICIENCY.

Dataset	Attack setting	FGSM	HGAttack	HGAttack-S	HSA	HSA-S	GUA	CD-ATTACK	RL-S2V	PR	GHAttack-D (Our)	GHAttack (Our)
ACM	White-box	31.02	347.22	n/a	45.62	n/a	0.04	0.65	2.05	0.60	0.51	0.68
	Gray-box	31.92	381.74	346.27	46.14	43.75	0.04	0.64	2.08	0.60	0.52	0.69
IMDB	White-box	37.69	379.23	n/a	55.05	n/a	0.04	1.07	3.39	0.96	0.67	1.02
	Gray-box	40.23	412.05	388.20	54.84	52.16	0.04	1.12	3.47	0.97	0.68	1.05
DBLP-Paper	White-box	215.42	2796.63	n/a	323.37	n/a	0.24	27.71	99.68	24.69	18.23	27.54
	Gray-box	220.30	2839.23	2732.82	326.08	304.62	0.25	28.23	102.74	24.20	18.64	27.72
DBLP-Author	White-box	50.05	586.25	n/a	76.20	n/a	0.05	6.08	21.18	5.36	1.54	6.10
	Gray-box	51.72	602.42	573.95	76.93	71.33	0.05	6.19	21.50	5.17	1.60	6.26
Freebase	White-box	n/a	3132.74	n/a	n/a	n/a	n/a	22.49	n/a	n/a	50.64	82.47
	Gray-box	n/a	3177.20	3110.25	n/a	n/a	n/a	23.86	n/a	n/a	50.93	84.02
Freebase-Noise	White-box	n/a	3143.38	n/a	n/a	n/a	n/a	22.72	n/a	n/a	51.20	82.29
	Gray-box	n/a	3182.62	3130.04	n/a	n/a	n/a	22.90	n/a	n/a	50.72	83.96

across heterogeneous relations still works, thus ensuring its superiority over the compared methods.

E. Attack Efficiency (Q4)

Here, we investigate the attack efficiency of all the methods. In particular, we are interested in the time cost of launching attacks during the inference phase. The model training required by the model-based methods can be done offline, thus we do not include this time cost in our evaluation. Table VI reports the time cost of different methods when conducting the previous experiments in Section VI-B (white-box) and Section VI-C (gray-box), where the mean AT in the white-box and the gray-box settings is computed in a similar way as the mean Micro-F1 and the mean Trans_{f_S} in Table III and Table IV, respectively. Due to the out-of-memory issues, we cannot report the results of FGSM, HSA, HSA-S, GUA, RL-S2V and PR on Freebase and Freebase-Noise. From the results, we have the following observations.

Model-based methods are much more time-efficient than optimization-based methods. In all the experiments, the mean AT of the model-based methods is significantly lower than that of the optimization-based methods, which demonstrates the advantage of model-based methods in launching fast and massive attacks. In contrast, due to the necessity of solving a time-consuming optimization problem for each target node, the optimization-based methods cannot deliver efficient attacks. For instance, HGAttack takes more than 40 hours to generate perturbations for the 13,000 target nodes in DBLP-Paper. FGSM, HSA and HSA-S are faster than HGAttack because of their simple optimization approaches, but they are still slower than the compared model-based methods.

GHAttack strikes a better balance between attack efficiency and attack effectiveness than the other model-based methods. In terms of attack speed, GUA is the fastest because of its simple attack scheme that only requires flipping the edge connections between a target node and a set of anchor nodes. However, as discussed in Section VI-B, this overly simplified attack scheme also limits the attack effectiveness of GUA. In addition, the mean AT of GHAttack-D, CD-ATTACK and PR is generally lower than that of GHAttack, as they only need to predict whether to perturb the candidate edges on a single relation, which are fewer in number than the candidate edges selected in GHAttack. Nevertheless, their inability to leverage graph heterogeneity makes them less effective than GHAttack. RL-SV2 is less time-efficient than the other model-based methods, as it requires performing ξ forward passes through the trained model to generate a perturbed graph.

VII. ETHICAL CONSIDERATION

The focus of this work is to reveal potential vulnerabilities in HGNNs rather than to promote their exploitation. We acknowledge the potential misuse of GHAttack and emphasize its responsible application under ethical AI principles. Our experiments are confined to publicly available datasets in controlled academic settings, avoiding harm to real HGNN-based systems. Moreover, the engineering efforts to attack real HGNN-based systems are substantial. We make it harder by not releasing our code publicly. We will, however, release our code including pre-trained checkpoints upon carefully considering each request. Researchers are urged to apply these techniques only in controlled environments, adhering to responsible disclosure protocols and ethical guidelines to prevent any potential misuse.

VIII. CONCLUSION

In this work, we propose a novel generative attack method called GHAttack for time-efficient and effective adversarial attacks on heterogeneous graph neural networks (HGNNs). The key idea of GHAttack is to train a perturbation generator to attack each target node by swiftly producing a perturbation via a forward pass. Meanwhile, GHAttack enables perturbations to modify edges across different relations of a heterogeneous graph, in order to deliver highly effective attacks. To achieve this, we design a novel model architecture for the generator, formulate its training as an optimization problem and efficiently solve it by addressing a series of technical challenges. We systematically evaluate the performance of GHAttack by conducting extensive experiments, where the results demonstrate the excellent attack efficiency and effectiveness of our method. In future work, we will explore the potential of GHAttack in attacking HGNNs in other tasks, such as link prediction and recommendation. In

addition, we will be committed to developing more effective defenses, such as building more robust HGNNs or designing effective indicators for monitoring attacks, to defend against GHAttack and other adversarial attacks.

REFERENCES

[1] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu, "Heterogeneous graph attention network," in *The World Wide Web Conference*, 2019, pp. 2022–2032.

[2] H. Hong, H. Guo, Y. Lin, X. Yang, Z. Li, and J. Ye, "An attention-based graph neural network for heterogeneous structural learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 4132–4139.

[3] J. Wei and X. Liao, "Dynamical threshold-based fractional anisotropic diffusion for speckle noise removal," *IEEE Transactions on Image Processing*, vol. 34, pp. 2826–2839, 2025.

[4] Z. Li, H. Liu, Z. Zhang, T. Liu, and N. N. Xiong, "Learning knowledge graph embedding with heterogeneous relation attention networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 8, pp. 3961–3973, 2021.

[5] Q. Liu, C. Long, J. Zhang, M. Xu, and D. Tao, "Aspect-aware graph attention network for heterogeneous information networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 5, pp. 7259–7266, 2022.

[6] G. Zhu, Z. Zhu, H. Chen, C. Yuan, and Y. Huang, "Hagnn: Hybrid aggregation for heterogeneous graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2024.

[7] Y. Sun, D. Zhu, H. Du, and Z. Tian, "Mhnf: Multi-hop heterogeneous neighborhood information fusion graph representation learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 7, pp. 7192–7205, 2022.

[8] Z. Hu, Y. Dong, K. Wang, and Y. Sun, "Heterogeneous graph transformer," in *Proceedings of The Web Conference*, 2020, pp. 2704–2710.

[9] C. Gao, Y. Zheng, N. Li, Y. Li, Y. Qin, J. Piao, Y. Quan, J. Chang, D. Jin, X. He *et al.*, "A survey of graph neural networks for recommender systems: Challenges, methods, and directions," *ACM Transactions on Recommender Systems*, vol. 1, no. 1, pp. 1–51, 2023.

[10] Q. Zhong, Y. Liu, X. Ao, B. Hu, J. Feng, J. Tang, and Q. He, "Financial defaulter detection on online credit payment via multi-view attributed heterogeneous information network," in *Proceedings of The Web Conference*, 2020, pp. 785–795.

[11] B. Hu, Z. Zhang, C. Shi, J. Zhou, X. Li, and Y. Qi, "Cash-out user detection based on attributed heterogeneous information network with a hierarchical attention mechanism," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 946–953.

[12] M. Zhang, X. Wang, M. Zhu, C. Shi, Z. Zhang, and J. Zhou, "Robust heterogeneous graph neural networks against adversarial attacks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 4, 2022, pp. 4363–4370.

[13] H. Zhao, Z. Zeng, Y. Wang, D. Ye, and C. Miao, "Hgattack: Transferable heterogeneous graph adversarial attack," in *2024 IEEE International Conference on Agents*, 2024, pp. 100–105.

[14] H. Li, J. Xu, L. Yin, Q. Wang, Y. Jiang, and J. Liu, "Metapath-free adversarial attacks against heterogeneous graph neural networks," *Information Sciences*, vol. 713, p. 122143, 2025.

[15] M. Salzmann *et al.*, "Learning transferable adversarial perturbations," *Advances in Neural Information Processing Systems*, vol. 34, pp. 13 950–13 962, 2021.

[16] H. Wu, C. Wang, Y. Tyshetskiy, A. Docherty, K. Lu, and L. Zhu, "Adversarial examples on graph data: Deep insights into attack and defense," *arXiv preprint arXiv:1903.01610*, 2019.

[17] B. Wang, B. Jiang, and C. Ding, "Fl-gnns: Robust network representation via feature learning guided graph neural networks," *IEEE Transactions on Network Science and Engineering*, vol. 11, no. 1, pp. 750–760, 2023.

[18] H. Chang, Y. Rong, T. Xu, W. Huang, H. Zhang, P. Cui, X. Wang, W. Zhu, and J. Huang, "Adversarial attack framework on graph embedding models with limited knowledge," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 5, pp. 4499–4513, 2022.

[19] Y. Zhu, Y. Lai, K. Zhao, X. Luo, M. Yuan, J. Wu, J. Ren, and K. Zhou, "From bi-level to one-level: A framework for structural attacks to graph anomaly detection," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 36, no. 4, pp. 6174–6187, 2024.

[20] X. Zang, Y. Xie, J. Chen, and B. Yuan, "Graph universal adversarial attacks: A few bad actors ruin graph learning models," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2021, pp. 3328–3334.

[21] H. Zhang, X. Yuan, C. Zhou, and S. Pan, "Projective ranking-based gnn evasion attacks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 8, pp. 8402–8416, 2023.

[22] J. Chen, D. Zhang, Z. Ming, K. Huang, W. Jiang, and C. Cui, "Graphattacker: A general multi-task graph attack framework," *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 2, pp. 577–595, 2021.

[23] J. Li, H. Zhang, Z. Han, Y. Rong, H. Cheng, and J. Huang, "Adversarial attack on community detection by hiding individuals," in *Proceedings of The Web Conference*, 2020, pp. 917–927.

[24] H. Dai, H. Li, T. Tian, X. Huang, L. Wang, J. Zhu, and L. Song, "Adversarial attack on graph structured data," in *International Conference on Machine Learning*, 2018, pp. 1115–1124.

[25] A. Liu, B. Li, T. Li, P. Zhou, and R. Wang, "An-gcn: an anonymous graph convolutional network against edge-perturbing attacks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 1, pp. 88–102, 2022.

[26] Y. Shang, Y. Zhang, J. Chen, D. Jin, and Y. Li, "Transferable structure-based adversarial attack of heterogeneous graph neural network," in *Proceedings of the ACM International Conference on Information and Knowledge Management*, 2023, pp. 2188–2197.

[27] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[28] J. Mu, B. Wang, Q. Li, K. Sun, M. Xu, and Z. Liu, "A hard label black-box adversarial attack against graph neural networks," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 108–125.

[29] X. Wang, H. Chang, B. Xie, T. Bian, S. Zhou, D. Wang, Z. Zhang, and W. Zhu, "Revisiting adversarial attacks on graph neural networks for graph classification," *IEEE Transactions on Knowledge and Data Engineering*, vol. 36, no. 5, pp. 2166–2178, 2023.

[30] S. Yun, M. Jeong, R. Kim, J. Kang, and H. J. Kim, "Graph transformer networks," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[31] Q. Lv, M. Ding, Q. Liu, Y. Chen, W. Feng, S. He, C. Zhou, J. Jiang, Y. Dong, and J. Tang, "Are we really making much progress? revisiting, benchmarking and refining heterogeneous graph neural networks," in *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2021, pp. 1150–1160.

[32] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *European Semantic Web Conference*, 2018, pp. 593–607.

[33] D. Zügner, O. Borchert, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on graph neural networks: Perturbations and their patterns," *ACM Transactions on Knowledge Discovery from Data*, vol. 14, no. 5, pp. 1–31, 2020.

[34] M. H. Kalos and P. A. Whitlock, *Monte carlo methods*, 2009.

[35] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," *arXiv preprint arXiv:1611.00712*, 2016.

[36] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1026–1034.

[37] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[38] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[39] J. Hestness, S. Narang, N. Ardalani, G. Diamos, H. Jun, H. Kianinejad, M. M. A. Patwary, Y. Yang, and Y. Zhou, "Deep learning scaling is predictable, empirically," *arXiv preprint arXiv:1712.00409*, 2017.

[40] H. Han, T. Zhao, C. Yang, H. Zhang, Y. Liu, X. Wang, and C. Shi, "Openhgnn: An open source toolkit for heterogeneous graph neural network," in *Proceedings of the ACM International Conference on Information and Knowledge Management*, 2022, pp. 3993–3997.

[41] Y. Zhang, S. Khan, and M. Coates, "Comparing and detecting adversarial attacks for graph deep learning," in *Proceedings of Representation Learning on Graphs and Manifolds Workshop*, 2019.