Stealing Training Graphs from Graph Neural Networks

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

Enyan Dai
enyandai@hkust-gz.edu.cn
Hong Kong University of Science and Technology (Guangzhou)
Guangzhou, China

Junjie Xu
junjiexu@psu.edu
The Pennsylvania State University
State College, USA

Jinyuan Jia
jinyuan@psu.edu
The Pennsylvania State University
State College, USA

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

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

ABSTRACT

Graph Neural Networks (GNNs) have shown promising results in modeling graphs in various tasks. The training of GNNs, especially on specialized tasks such as bioinformatics, demands extensive expert annotations, which are expensive and usually contain sensitive information of data providers. The trained GNN models are often shared for deployment in the real world. As neural networks can memorize the training samples, the model parameters of GNNs have a high risk of leaking private training data. Our theoretical analysis shows the strong connections between trained GNN parameters and the training graphs used, confirming the training graph leakage issue. However, explorations into training data leakage from trained GNNs are rather limited. Therefore, we investigate a novel problem of stealing graphs from trained GNNs. To obtain high-quality graphs that resemble the target training set, a graph diffusion model with diffusion noise optimization is deployed as a graph generator. Furthermore, we propose a selection method that effectively leverages GNN model parameters to identify training graphs from samples generated by the graph diffusion model. Extensive experiments on real-world datasets demonstrate the effectiveness of the proposed framework in stealing training graphs from the trained GNN. The code is publicly available at https://github.com/ventr1c/GraphSteal.

CCS CONCEPTS

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

KEYWORDS

Graph Neural Networks; Graph Stealing Attack; Privacy

ACM Reference Format:
Minhua Lin, Enyan Dai, Junjie Xu, Jinyuan Jia, Xiang Zhang, and Suhang Wang. 2025. Stealing Training Graphs from Graph Neural Networks. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.1 (KDD '25), August 3–7, 2025, Toronto, ON, Canada.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3690624.3709289

KDD '25, August 3–7, 2025, Toronto, ON, Canada

ACM ISBN 979-8-4007-1245-6/25/08
https://doi.org/10.1145/3690624.3709289

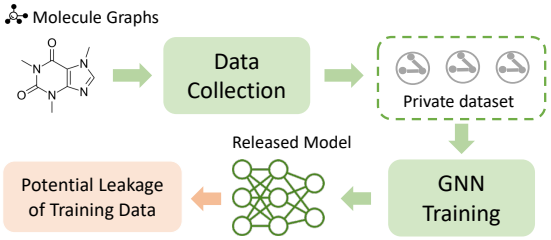


Figure 1: Illustration of GNNs training and releasing.

1 INTRODUCTION

Graph-structured data pervade numerous real-world applications such as social networks [24], finance systems [62], and molecular graphs [63]. Graph Neural Networks (GNNs) have shown promising results in modeling graphs by adopting a message passing scheme [35, 41, 61, 70], which updates a node's representation by aggregating information from its neighbors. The learned representation can preserve both node attributes and local graph structural information, facilitating various tasks, such as node classification [] and graph classifications [18].

As shown in Fig 1, generally, abundant training data is required to train a high-performance GNN model. This becomes particularly expensive for critical applications such as molecule property prediction, which demands expert annotations. Moreover, the training data may hold sensitive information belonging to its providers. Consequently, protecting the privacy of the training data is imperative. After training on the private training data, the trained GNN model is often released for downstream applications. For example, a well-trained molecule property predictor may be made open source, supporting the direct deployment for the designed task or model initialization for other tasks. However, neural networks can memorize the training data, even when they have great generalization ability [77]. It is also demonstrated that the trained parameters of MLP are linear combinations of the derivatives of the network at a set of training data points [49], causing potential private data leakage. Our theoretical analysis in Theorem 4.1 further shows that the above observations can be extended to GNNs. Hence, releasing GNN models potentially threatens the privacy of the private training data. Therefore, in this paper, we investigate a novel problem of

stealing training graphs from trained GNN models when the model architecture and parameters are known/released.

Several initial efforts are made in model inversion attacks, which aim to reconstruct graph topologies [79] or infer the sensitive node attributes attributes [16]. However, they are proposed to reconstruct the train graph/node from node/graph embeddings, which cannot be applied for training graph stealing as embeddings of private training data are unavailable for attackers. Recently, GraphMI [82] proposes to reconstruct the adjacency matrix of the training graph in a white-box setting where the trained model parameters are available. However, GraphMI requires all the node attributes of the training graph to be available for topology reconstruction. This assumption is impractical in real-world applications. Therefore, in this paper, we propose to steal the training graphs from the trained GNN model without any information of the training data, which is not explored by existing works.

However, it is non-trivial to perform the graph stealing attack. There are two major challenges to be addressed. *First*, it is challenging to ensure both the realism and quality of graphs reconstructed from the trained GNN model. Existing model inversion attacks on GNNs mainly focus at the node level, concentrating on reconstructing either links [25, 82] or node attributes [16, 79] of the target graphs. However, this focus is inadequate for graph stealing attacks because graph-structured data comprises both graph topology and node attributes, rendering these methods insufficient for comprehensive graph-level reconstruction. Our empirical analysis in Sec. 5.2.1 further demonstrates that current methods encounter difficulties in providing realistic and high-quality graphs during graph-level reconstruction. *Second*, how to effectively utilize the trained GNN model parameters to extract the training graph information? Previous works on model inversion attacks on GNNs [16, 25, 82] typically rely on comprehensive side information of target samples for graph reconstruction. The investigations on extracting training data information from trained GNN parameters without any partial training data information poses a new challenge.

In an effort to address the aforementioned challenges, we propose a novel *Graph Steal*ing attack framework (GraphSteal). Specifically, to overcome the challenge of reconstructing high-quality graphs, we employ a graph diffusion model as the graph generator. In addition, this graph diffusion model used in GraphSteal adopts a diffusion noise optimization algorithm, which can produce a set of candidate graphs that more closely resemble the training set of the target GNN. Moreover, according to our theoretical analysis in Theorem 4.1, there is a strong connection between the GNN model parameters and training data. Based on the theorem, we propose a model parameter-guided graph selection method, which leverages the parameters of GNNs to identify the training graphs from the candidate graph set generated by the graph diffusion model. In summary, our main contributions are:

- We study a new problem of stealing training graphs from a trained GNN without any partial information on training data;
- We propose GraphSteal, a novel attack framework that can effectively recreate high-quality graphs that are part of the target training dataset by leveraging parameters of GNN models; and
- Extensive experiments on various real-world datasets demonstrate the effectiveness of our proposed GraphSteal in accurately reconstructing private training graphs of trained GNN models.

2 RELATED WORK

Graph Neural Networks. Graph Neural Networks (GNNs) [32, 35, 37, 45, 47, 69, 75] have shown great power in modeling graph-structured data, which have been deployed to various applications such as social network analysis [8, 19, 43, 46, 81], drug discovery [7, 9, 11, 71, 80] and energy network analysis [4]. The success of GNNs lies in the message-passing mechanism, which iteratively aggregates a node's neighborhood information to refine the node's representations. For example, GCN [35] combines a node's neighborhood information by averaging their representations with the target center nodes. To improve the expressivity of GNNs, GIN [70] further incorporates a hidden layer in combining the neighbors' information. Inspired by the success of transformers in modeling image and text [13, 14, 20, 22, 44], graph transformer is also proposed [21, 42, 51, 57, 74, 78]. Generally, graph transformer has a global attention mechanism for graph embedding learning. It shows promising results, especially in molecule property prediction. Despite the great achievements, GNNs could be vulnerable to privacy attacks, which largely constrain their adoption in safety-critical domains such as bioinformatics and financial analysis.

Privacy Attacks on GNNs. The training of GNNs requires a large amount of data. In critical domains such as bioinformatics [38] and healthcare [39], sensitive data of users will be collected to train a powerful GNN to facilitate the services. However, recent studies show that privacy attacks can extract various private information from GNN models [5, 6, 10, 12, 16, 31, 40, 53], threatening the privacy of users. For example, membership inference attacks [16, 26, 53, 67] can identify whether a target sample is in the training set. This is achieved by learning a binary classifier on patterns such as posteriors that training and test samples exhibit different distributions. Model extract attacks [65] aim to steal the target model by building a model that behaves similarly to the target model. Model inversion attacks, also known as reconstruction attacks, try to infer the private information of the test/training data. For example, Duddu et al. [16] propose to infer sensitive attributes of users from their node embeddings. The reconstruction of the adjacency matrix from node embeddings is studied in [79]. GraphMI [82] further considers a white-box setting for adjacency matrix reconstruction, where the target GNN model parameters and node features are available for attackers. Our GraphSteal lies in the model inversion attack. However, GraphSteal is inherently different from the aforementioned methods because (**i**) we focus on a new problem of stealing training graphs from the trained GNN without even partial information of training set; (**ii**) we propose a novel framework GraphSteal which designs a diffusion noise optimization algorithm and a model parameter-guided graph selection mechanism to reconstruct high-quality training graphs of the target GNN model.

Graph Diffusion Models. Diffusion models have showcased their exceptional performance on various tasks, including image generation [29, 58], text-to-image generation [2, 56] and video generation [28]. Generally, a diffusion model has two phases: (i) the diffusion phase, where noise is incrementally added to the clean data; and (ii) the denoising phase, where a model learns to predict and remove the noise. After trained, the denoise module can effectively generate realistic samples with noises as input. Recent works

try to extend diffusion models to graphs. For example, Vignac et al. [60] proposes DiGress to bridge the gap between the discreteness of graph structure and the continuity of the diffusion and denoising process. Jo et al. [34] uses a system of stochastic differential equations to model the joint distribution of nodes and edges. Luo et al. [48] conduct the graph diffusion process from the spectral domain. EDM [30] and GeoDiff [72] aim to preserve the equivariant information of 3D graphs.

3 BACKGROUNDS AND PRELIMINARIES

Notations. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ denote an attributed graph, where $\mathcal{V} = \{v_1, \ldots, v_n\}$ is the set of n nodes, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges, and $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$ is the set of node attributes with $\mathbf{x}_i$ being the node attribute of v_i. $\mathbf{A} \in \mathbb{R}^{n \times n}$ is the adjacency matrix of $\mathcal{G}$, where $\mathbf{A}_{ij} = 1$ if nodes v_i and v_j are connected; otherwise $\mathbf{A}_{ij} = 0$. In this paper, we focus on graph stealing attack from a GNN classifier trained on a private labeled graph dataset $\mathcal{D}_L = \{\mathcal{G}_1, \ldots, \mathcal{G}_l\}$ with $\mathcal{Y}_L = \{y_1, \ldots, y_l\}$ being the corresponding labels. $y_i \in \{1, 2, \ldots, C\}$ is the label of $\mathcal{G}_i$, where C is the number of classes.

3.1 Graph Diffusion Model

Graph diffusion models have demonstrated a robust capability in generating realistic graphs. In this paper, we mainly employ DiGress [60], a popular discrete denoising graph diffusion model, as the graph generator. The diffusion process of DiGress is a Markov process consisting of successive graphs edits (edge addition or deletion, node or edge category edit) that can occur independently on each node or edge. As the nodes and edges are considered to belong to one of the given categories, the noises are modeled as transition matrices $(Q^1, \ldots, Q^T)$, where $\left[Q^t\right]_{ij}$ is the probability of jumping from category i to category j. To invert this diffusion process, it trains a graph transformer network to predict the clean graph from a noisy input. Specifically, DiGress diffuses each node and edge feature separately by applying transition matrices. The diffusion process at t-th step can treated as sampling node type and edge type of $\mathcal{G}^t$ from the categorical distributions $\mathbf{X}^{t-1}Q^t_{\mathbf{X}}$ and $\mathcal{E}^{t-1}Q^t_{\mathcal{E}}$, respectively, which can be written as

$$q(\mathcal{G}^t \mid \mathcal{G}^{t-1}) = (\mathbf{X}^{t-1}Q^t_{\mathbf{X}}, \mathcal{E}^{t-1}Q^t_{\mathcal{E}}), \tag{1}$$

where $\mathcal{G}^t$ is the noisy graph at t-th diffusion step with $\mathcal{G}^0$ being the original clean graph and q is the noise model. $\mathbf{Q}^t_{\mathbf{X}}$ and $\mathbf{Q}^t_{\mathcal{E}}$ are the transition matrices for $\mathbf{X}$ and $\mathcal{E}$, respectively. Following [60], with Eq.(1), the diffusion from $\mathcal{G}_0$ to $\mathcal{G}_t$ can be written as

$$q(\mathcal{G}^t \mid \mathcal{G}^0) = (\mathbf{X}\overline{Q}^t_{\mathbf{X}}, \mathcal{E}\overline{Q}^t_{\mathcal{E}}), \tag{2}$$

where $\overline{Q}^t_{\mathbf{X}} = Q^1_{\mathbf{X}} \ldots Q^t_{\mathbf{X}}$ and $\overline{Q}^t_{\mathcal{E}} = Q^1_{\mathbf{X}} \ldots Q^t_{\mathcal{E}}$.

In the denoising process, DiGress learns a denoising neural network R_ϕ parameterized by ϕ to predict the clean graph $\mathcal{G}^0$ from the noisy graph $\mathcal{G}^t$. The denoising process is expressed as:

$$p_\phi(\mathcal{G}^{t-1} \mid \mathcal{G}^t) = \prod_{1 \le i \le n} p_\phi(\mathbf{x}_i^{t-1} \mid \mathcal{G}^t) \prod_{1 \le i,j \le n} p_\phi(e_{ij}^{t-1} \mid \mathcal{G}^t), \tag{3}$$

where $p_\phi(\mathbf{x}_i^{t-1} \mid \mathcal{G}^t)$ denotes the probability of predicting $\mathbf{x}_i^{t-1}$ from $\mathcal{G}^t$ and $p_\phi(e_{ij}^{t-1} \mid \mathcal{G}^t)$ denotes the probability of predicting e_{ij}^{t-1} from $\mathcal{G}^t$, where e_{ij} is the edge category between node i and node j, For simplicity, we define R_ϕ parameterized by ϕ as the function to denoise $\mathcal{G}_i$ and then get the next sampled value as

$$\mathcal{G}_i^{t-1} = R_\phi(\mathcal{G}_i^t, t) \tag{4}$$

DiGress is trained to revert the diffusion process to recover the clean graph, which equips the denoising network with the ability of generating realistic graphs from input noises.

3.2 Threat Model

3.2.1 Attacker's Goal. Given a target GNN classifier $f_{\boldsymbol{\theta}}$ trained on a private target graph dataset $\mathcal{D}_t = \{\mathcal{G}_1, \ldots, \mathcal{G}_{|\mathcal{D}_t|}\}$, the goal of the adversary in graph stealing attack is to reconstruct graphs in $\mathcal{D}_t$ from the target model $f_{\boldsymbol{\theta}}$.

3.2.2 Attacker's Knowledge and Capability. We focus on the graph stealing attacks in the white-box setting. The information of the target model $f_{\boldsymbol{\theta}}$ including model architecture and model parameters $\boldsymbol{\theta}$ is available to the attacker. This is a reasonable setting in real-world as model providers will often release the trained model to customers for downstream applications. Moreover, we consider a practical setting that the attacker does not have access to any sensitive information of the target dataset $\mathcal{D}_t$, including graph topology $\mathbf{A}_i$, node attributes $\mathbf{X}_i$ and graph label y_i of $\mathcal{G}_i \in \mathcal{D}_t$ as the private data could contain sensitive information or is intellectual property of the data owner. However, we assume that an auxiliary dataset $\mathcal{D}_a = \{\mathcal{G}_i, y_i\}_{i=1}^{|\mathcal{D}_a|}$, which shares a similar low dimensional manifold [15] as that of $\mathcal{D}_t$, is available to the attackers. This assumption is reasonable because in realistic scenarios, e.g., drug design [60], the attacker has access to many publicly available molecules and has high motivation to steal private molecules that are easy to synthesize and have high activity on specific targets.

3.3 Problem Definition

With the above notations and the description of graph stealing attack in Sec. 3.2, the objective of graph stealing attack is to reconstruct graphs in the target dataset $\mathcal{D}_t$ that are used to train the target GNN model without any partial information of $\mathcal{D}_t$. This problem can be formally defined as:

Problem 1 (Graph Stealing Attack)**.** *Given a target GNN classifier: $f_{\boldsymbol{\theta}} : \mathcal{G} \rightarrow \mathbf{y}$ trained on a privately owned dataset $\mathcal{D}_t = \{(\mathcal{G}_i, y_i)\}_{i=1}^{|\mathcal{D}_t|}$, where $\mathbf{y}_i \in \{1, 2, \ldots, C\}$ is $\mathcal{G}_i$'s label, and an auxiliary dataset $\mathcal{D}_a$ that share the same manifold characteristics as $\mathcal{D}_t$ with $\mathcal{D}_a \cap \mathcal{D}_t = \emptyset$, we aim to extract a subset of private training data $\mathcal{D}_s \subset \mathcal{D}_t$. The architecture and model parameters of $f_{\boldsymbol{\theta}}$ are accessible.*

4 METHODOLOGY

In this section, we present the details of our GraphSteal, which aims to reconstruct the training graphs from the target GNN model without even partial information of the target dataset. There are mainly two challenges to be addressed for achieving better reconstruction performance: **(i)** how to ensure both the realism and quality of graphs reconstructed from the trained GNN model; and **(ii)** how to utilize the trained GNN model parameters to extract the training graph information. To address these challenges, we propose a novel framework GraphSteal, which is illustrated in Fig. 2. GraphSteal is

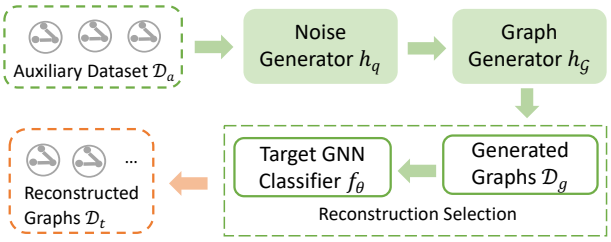


Figure 2: An overview of proposed GraphSteal.

composed of a graph generator $h_\mathcal{G}$, a noise generator h_q, a reconstructed graph selector h_s and the target GNN model f_θ. Specifically, a graph diffusion model is adopted as the graph generator $h_\mathcal{G}$ to generate realistic and high-quality graphs. The noise generator h_q takes graphs from the auxiliary dataset $\mathcal{D}_a$ as inputs to learn input noises for the graph generator $h_\mathcal{G}$, aiming to ensure $h_\mathcal{G}$ can generate graphs $\mathcal{D}_g$ mimic to the graphs within the target dataset $\mathcal{D}_t$. $\mathcal{D}_t$ is the training set of f_θ. Finally, we perform a reconstruction selection by leveraging parameters θ of the target classifier f_θ to select the top-k most representative graphs from $\mathcal{D}_g$ that closely resemble the target dataset $\mathcal{D}_t$. Next, we give the detailed design of the proposed framework.

4.1 Reconstruction Generation

In this subsection, we give the details of reconstruction generation. We first train a graph diffusion model as the graph generator to guarantee the quality of the reconstructions in Sec. 4.1.1. We then propose to devise input noises for the graph diffusion model by solving an optimization problem in Sec. 4.1.2. Finally, we utilize the optimized input noises to produce the reconstructions in Sec. 4.1.3.

4.1.1 Building Graph Generator. To ensure that we can reconstruct the high-quality and realistic training graphs from trained GNN model, one promising direction is to utilize graph diffusion models that have powerful generation capability to generate graphs. Graph diffusion models are usually trained on a specific dataset to learn its distribution. The trained graph diffusion model can then generate diverse realistic samples that are representatives of the learned data distribution. As the auxiliary dataset $\mathcal{D}_a$ share a similar distribution as the target dataset $\mathcal{D}_t$, we first train a graph diffusion model $h_\mathcal{G}$ on $\mathcal{D}_a$ to learn the underlying data distribution. After $h_\mathcal{G}$ is trained, it can generate high-quality graphs that are likely to follow the distribution of $\mathcal{D}_t$. We adopt DiGress introduced in Sec. 3.1 as the graph diffusion model due to its effectiveness.

4.1.2 Diffusion Noise Optimization. Though we can use $h_\mathcal{G}$ to generate realistic graphs, there are two issues: (i) If we use random noise as input to $h_\mathcal{G}$ to generate graphs, the chance of hitting the training graph in $\mathcal{D}_t$ is very low as the noise input space is very large; and (ii) Though the auxiliary dataset $\mathcal{D}_a$ shares a similar low dimensional manifold with $\mathcal{D}_t$, they might have a slightly distribution shift from each other. Hence, directly applying $h_\mathcal{G}$ trained on $\mathcal{D}_a$ to generate the reconstructed graphs may still not accurately represent the graphs in $\mathcal{D}_t$ due to the distribution shift.

To address this issue, inspired by [3, 50], we propose to first find input noises that are likely to results in graphs closely resembling those in $\mathcal{D}_t$, which reformulates the problem of generating training graphs in $\mathcal{D}_t$ as noise optimization problem, i.e., how to design the input noise $\mathcal{G}'$ for the graph diffusion model $h_\mathcal{G}$ such that $h_\mathcal{G}$ can generate graphs closely resembling those in $\mathcal{D}_t$?

To solve the above problem, we propose a novel diffusion noise generator to learn the input noise to guide the generation of the graph diffusion model to generate graphs better represent those in the target dataset. Our intuition is that given a graph $\mathcal{G}_i$ with the ground-truth label y_i , if $\mathcal{G}_i$ has higher probability to be predicted to the target class y_i by the GNN model f_θ, $\mathcal{G}_i$ is more likely to be the training graph of f_θ. Moreover, as the auxiliary dataset is available to the attackers, to improve the efficiency of learning input noise, we can then select graphs with higher prediction scores as the input noises to guide the graph diffusion model in generating graphs with similar characteristics to those in target dataset [50].

Therefore, we first use a metric to measure the prediction score of the graphs with the target model f_θ for selecting graphs from $\mathcal{D}_a$ as the input noises for the graph diffusion model $h_\mathcal{G}$. Formally, given a graph $\mathcal{G}_i \in \mathcal{D}_a$, the metric score is defined as:

$$s(\mathcal{G}_i, y_i; f_\theta) = f_\theta(\mathcal{G}_i)_{y_i} \tag{5}$$

where y_i denotes the ground-truth label of $\mathcal{G}_i$. After getting the metric score of each graph in $\mathcal{D}_a$, we select graphs with the top-m highest score in each class. The set of the selected graphs are denoted as $\mathcal{D}_c = \{\mathcal{G}_1, \ldots, \mathcal{G}_M\}$, where $M = C \cdot m$.

To further ensure the similarity of selected graphs to the target graphs, thereby improving reconstruction performance, we treat the adjacency matrix $\mathbf{A}_i$ and the node attributes $\mathbf{X}_i$ of the graph $\mathcal{G}_i \in \mathcal{D}_c$ as variables to be optimized. Specifically, they are optimized by minimizing the loss between the prediction $f_\theta(\mathcal{G}_i)$ and y_i as:

$$\sum_{i=1}^{M} \min_{\mathcal{G}_i \in \mathcal{D}_c} \mathcal{L}(f_\theta(\mathcal{G}_i), y_i), \tag{6}$$

where $\mathcal{L}$ is the training loss (e.g., cross-entropy loss) of f_θ. The optimized graphs are then denoted as $\mathcal{D}_c' = \{\mathcal{G}_1, \ldots, \mathcal{G}_M\}$

4.1.3 Generating Graphs. After generating optimized graphs based on Sec. 4.1.2, one straightforward way of reconstruction is to use these optimized graphs directly as the reconstructed ones. However, such optimized graphs often fall short in terms of realism and validity when applied to real-world scenarios, rendering the reconstruction process ineffective. This shortfall is further evidenced by the experimental results in Sec. 5.4, which show the unrealistic nature and invalidity of these optimized graphs. To address this issue, inspired by [50, 73], we propose to apply SDEdit [50] to generate graphs to enhance the generation quality and the resemblance of the generated graphs to those within the target dataset. The key idea of SDEdit is to "hijack" the generative process of the graph diffusion $h_\mathcal{G}$ by adding a suitable amount of noise to smooth out the undesirable details. This noise addition aims to blur out unwanted details while preserving the key structure of the input graphs. The noised graphs are then fed back into $h_\mathcal{G}$ to progressively eliminate the noise, yielding denoised outputs that are both realistic and closely resemble the graphs in the target dataset $\mathcal{D}_t$. Specifically, given an input noise $\mathcal{G}_i \in \mathcal{D}_c'$ optimized in Sec. 4.1.2, we first diffuse $\mathcal{G}_i$ for K steps to obtain $\mathcal{G}_i^K$ using diffusing module of $h_\mathcal{G}$ as

$$\mathcal{G}_i^K \sim q(\mathcal{G}_i^K | \mathcal{G}_i) \tag{7}$$

where $q(\cdot|\cdot)$ is the noise function in Eq. (2). We then run the reverse denoising process R_ϕ in the graph diffusion model $h_\mathcal{G}$ according to

Eq. (3) in Sec. 3.1 as

$$h_{\mathcal{G}}(\mathcal{G}_i) = R_\phi(\dots R_\phi(R_\phi(\mathcal{G}_i^K), K-1)\dots, 0), \tag{8}$$

It bridges the gap between the input data distribution (which is usually close to the target data distribution) and the auxiliary data distribution, ensuring that the generated graphs are realistic and closely similar to the target graphs in $\mathcal{D}_t$. The set of the generated graphs are denoted as $\mathcal{D}_g = \{\mathcal{G}_1, \dots, \mathcal{G}_M\}$.

4.2 Reconstruction Selection

With the above process in Sec. 4.1, we can get a collection of high-quality labeled graphs that follow a similar distribution with the target dataset $\mathcal{D}_t$. However, it is possible that some generated graphs, while resembling the distribution of $\mathcal{D}_t$, do not precisely match the graphs in $\mathcal{D}_t$. Thus, to further refine the performance of our graph stealing attack and enhance the accuracy of our reconstructions, we propose a novel model parameter-guided graph selection method to select the most representative samples among the generated graphs, prioritizing those that most closely approximate the actual graphs in $\mathcal{D}_t$. Our major intuition comes from the strong correlation between the model parameters and the training data throughout the training process of neural networks [33, 49]. Building upon this insight, we leverage the parameters $\boldsymbol{\theta}$ of the target GNN classifier $f_{\boldsymbol{\theta}}$ to select the graphs that most closely resemble the target training graphs of $f_{\boldsymbol{\theta}}$. Specifically, we first introduce the connection between model parameters and the training data in Sec. 4.2.1. Then, we present a novel approach based on the above connection to select samples from $\mathcal{D}_g$ that most closely resemble to graphs in the target dataset $\mathcal{D}_t$ in Sec. 4.2.2.

4.2.1 Connections between Model Parameters and Training Data. We begin by formally defining homogeneous neural networks:

Definition 4.1 (Homogeneous Neural Networks [49])**.** *Let f be a neural network with model parameters as $\boldsymbol{\theta}$. Then f is a homogeneous neural network if there is a number $L > 0$ such that the model output $f(\mathcal{G}; \boldsymbol{\theta})$ satisfies the following equation:*

$$f(\mathcal{G}; \sigma\boldsymbol{\theta}) = \sigma^L f(\mathcal{G}; \boldsymbol{\theta}),\ \forall \sigma > 0 \tag{9}$$

Note that essentially any message-passing based GNNs (e.g. GCN [35] and SGC [66]) with ReLU activations is homogeneous w.r.t the parameters $\boldsymbol{\theta}$ if it does not have any skip-connections (e.g., GraphSage [24]) or bias terms, except possibly for the first layer. More details of the proof and discussion is in Appendix A.2.

We then present Lemma 4.1 below to show the connection between model parameters and the training data of homogeneous neural networks. Essentially, the lemma shows that the process of training GNNs with gradient descent or gradient flow can be formulated as a constrained optimization problem related to margin maximization, which is formally given as:

Lemma 4.1 ([49])**.** *Let $f_{\boldsymbol{\theta}}$ be a homogeneous ReLU neural network with model parameters $\boldsymbol{\theta}$. Let $\mathcal{D}_t = \{(\mathcal{G}_i, y_i)\}_{i=1}^{|\mathcal{D}_t|}$ be the classification training dataset of $f_{\boldsymbol{\theta}}$, where $y_i \in \{1, \dots, C\}$. Assume $f_{\boldsymbol{\theta}}$ is trained by minimizing the cross entropy loss $\mathcal{L}_{ce} = \sum_{i=1}^{|\mathcal{D}_t|} \log(1 + \sum_{j \neq y_i} e^{-q_{ij}(\boldsymbol{\theta})})$ over $\mathcal{D}_t$ using gradient flow, where $q_{ij}(\boldsymbol{\theta}) = f_{\boldsymbol{\theta}}(\mathcal{G}_i)_{y_i} - f_{\boldsymbol{\theta}}(\mathcal{G}_i)_j$ and $f_{\boldsymbol{\theta}}(\mathcal{G}_i)_{y_i}$ denotes the y_i-th entry of the logit score vector $f_{\boldsymbol{\theta}}(\mathcal{G}_i)$ before applying the softmax normalization. Then, gradient flow converges in direction towards a first-order stationary point of the following max-margin optimization problem:*

$$\min_{\boldsymbol{\theta}} \frac{1}{2}||\boldsymbol{\theta}||_2^2, \quad s.t. \quad q_{ij}(\boldsymbol{\theta}) \geq 1,\ \forall i \in [|\mathcal{D}_t|], j \in [C] \backslash \{y_i\}. \tag{10}$$

Lemma 4.1 guarantees directional convergence to a *Karush-Kuhn-Tucker* point (or *KKT* point). More details of KKT points can be found in Appendix A.1.

As described in Sec. 4.2.1, we can extend Lemma 4.1 to homogeneous GNNs. Formally, given a target GNN classifier $f_{\boldsymbol{\theta}}$ that satisfy Eq. (9) in Def. 4.1, the formal theorem is presented as follows:

Theorem 4.1. *Let $f_{\boldsymbol{\theta}}$ be the target GNN classifier that preserves the homogeneity in Def. 4.1. $f_{\boldsymbol{\theta}}$ is trained on the labeled set $\mathcal{D}_t = \{(\mathcal{G}_i, y_i)\}_{i=1}^{|\mathcal{D}_t|}$, where $y_i \in \{1, \dots, C\}$. After convergence of training $f_{\boldsymbol{\theta}}$ using gradient descent, the converged model parameter $\tilde{\boldsymbol{\theta}}$ satisfies the following equation:*

$$\tilde{\boldsymbol{\theta}} = \sum_{i=1}^{|\mathcal{D}_t|} \lambda_i (\nabla_{\boldsymbol{\theta}} f_{\tilde{\boldsymbol{\theta}}}(\mathcal{G}_i)_{y_i} - \nabla_{\boldsymbol{\theta}} \max_{j \neq y_i} \{f_{\tilde{\boldsymbol{\theta}}}(\mathcal{G}_i)_j\}), \tag{11}$$

where

$$\beta_i(\tilde{\boldsymbol{\theta}}) = f_{\tilde{\boldsymbol{\theta}}}(\mathcal{G}_i)_{y_i} - \max_{j \neq y_i} \{f_{\tilde{\boldsymbol{\theta}}}(\mathcal{G}_i)_j\} \geq 1, \forall i \in |\mathcal{D}_t|, \tag{12}$$

$$\lambda_1, \dots, \lambda_{|\mathcal{D}_t|} \geq 0, \tag{13}$$

$$\lambda_i = 0 \quad \text{if } \beta_i(\tilde{\boldsymbol{\theta}}) \neq 1, \forall i \in |\mathcal{D}_t|. \tag{14}$$

The proof is in Appendix A.3. Eq. (11) implies that the parameters $\tilde{\boldsymbol{\theta}}$ are a linear combination of the derivatives of the target GNN $f_{\boldsymbol{\theta}}$ at the training graphs $\mathcal{G}_i \in \mathcal{D}_t$. Especially, according to Eq. (14), it is possible that $\lambda_i = 0$ for some training graph $\mathcal{G}_i \in \mathcal{D}_t$, which implies that such $\mathcal{G}_i$ has no impact on the final model parameters of $f_{\boldsymbol{\theta}}$. Thus, to recover training dataset $\mathcal{D}_t$ from $f_{\boldsymbol{\theta}}$, it is natural to identify graphs $\{\mathcal{G}_1, \dots, \mathcal{G}_k\}$ from $\mathcal{D}_g$ with $\lambda_i > 0$ to satisfy Eq. (11), which means they contribute to $\boldsymbol{\theta}$.

4.2.2 Optimizing Selection Mask. Based on Theorem 4.1, the selection of the reconstructed graphs can be reformulated as finding optimal graph selection masks $\Lambda = \{\lambda_1, \dots, \lambda_M\}$ for the set of generated graphs $\mathcal{D}_g = \{\mathcal{G}_1, \dots, \mathcal{G}_M\}$ in Sec. 4.1.3. Specifically, our objective is to optimize Λ to select graphs from $\mathcal{D}_g$ to closely approximate the model parameters $\boldsymbol{\theta}$ for $f_{\boldsymbol{\theta}}$. The objective function for selection can be written as:

$$\min_{\Lambda} \mathcal{L}_s(\mathcal{D}_g, \Lambda, \boldsymbol{\theta}) = \left\| \boldsymbol{\theta} - \sum_{i=1}^{M} \lambda_i \left(\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathcal{G}_i)_{y_i} - \nabla_{\boldsymbol{\theta}} \max_{j \neq y_i} \{f_{\boldsymbol{\theta}}(\mathcal{G}_i)_j\} \right) \right\|_2^2, \quad \text{s.t.} \quad \lambda_i \geq 0, \quad \forall i \in \{1, \dots, M\} \tag{15}$$

where $\boldsymbol{\theta}$ and $\mathcal{G}_i \in \mathcal{D}_g$ are known and $\{\lambda_1, \dots, \lambda_M\}$ is the set of graph selection mask to be optimized. λ_i can be view as the metric score of selecting $\mathcal{G}_i$ as the final reconstructed graphs. Since this is a constrained optimization problem, to fulfill the constraint in Eq. (15), we introduce a constraint loss $\mathcal{L}_\lambda(\Lambda)$, which is defined as:

$$\mathcal{L}_\lambda(\Lambda) = \sum\nolimits_{i=1}^{M} \max\{-\lambda_i, 0\}. \tag{16}$$

Therefore, the final objective function for selection is:

$$\min_{\Lambda} \mathcal{L}_s(\mathcal{D}_g, \Lambda, \boldsymbol{\theta}) + \alpha \mathcal{L}_\lambda(\Lambda), \tag{17}$$

Algorithm 1 Reconstruction Algorithm of GraphSteal

Input: Auxiliary dataset $\mathcal{D}_a$, trained target GNN classifier $f_{\boldsymbol{\theta}}$ with parameters $\boldsymbol{\theta}$, hyperparameter α and the selection number k
Output: Reconstructed graphs set $\mathcal{D}_r$
1: Train a graph generator $h_{\mathcal{G}}$ on $\mathcal{D}_a$
2: Select input graphs from $\mathcal{D}_a$ based on Eq. (5) and obtain $\mathcal{D}_c$
3: Optimize $\mathcal{D}_c$ to get $\mathcal{D}_c'$ based on Eq. (6)
4: Generate graphs $\mathcal{D}_g$ based on Eq. (8)
5: **while** not converged **do**
6: Calculate $\mathcal{L}_s(\mathcal{D}_g, \Lambda, \boldsymbol{\theta})$ based on Eq. (15)
7: Calculate $\mathcal{L}_\lambda(\Lambda)$ based on Eq. (16).
8: Update the selection masks Λ with gradient decent on $\nabla_\Lambda(\mathcal{L}_s + \alpha\mathcal{L}_\lambda)$ based on Eq. (17).
9: **end while**
10: Select graphs with the top-k highest value in Λ as $\mathcal{D}_r$

where α is the hyperparameter to balance the contribution of $\mathcal{L}_\lambda(\Lambda)$. After $\mathcal{L}_\lambda(\Lambda)$ is learned, we select graphs with the top-k highest scores as the final reconstructed graphs, which are denoted as $\mathcal{D}_r = \{\mathcal{G}_1, \ldots, \mathcal{G}_k\}$.

4.3 Reconstruction Algorithm

The reconstruction algorithm of GraphSteal is in Alg. 10. Specifically, we train a graph diffusion model $h_{\mathcal{G}}$ as the graph generator (line 1), and select graphs from the auxiliary dataset $\mathcal{D}_a$ as the input graphs $\mathcal{D}_c$ (line 2). $\mathcal{D}_c$ is then optimized based on Eq. (6) (line 3) to use to generate $\mathcal{D}_g$ based on Eq. (8) (line 4). From line 5 to line 8, we learn the graph selection masks Λ based on Eq. (15). Finally, we select graphs with the top-k highest value in Λ as $\mathcal{D}_r$ (line 9).

5 EXPERIMENTS

In this section, we conduct experiments on various real-world datasets to answer the following research questions: (i) **Q1**: Can GraphSteal effectively reconstruct private training graphs of GNNs by leveraging the parameters of GNNs? (ii) **Q2**: How do the number of final selected graphs affect the performance of reconstruction? (iii) **Q3**: How does each component of GraphSteal contribute to the effectiveness in stealing training graphs?

5.1 Experimental Setup

5.1.1 Datasets. We conduct experiments on 3 public real-world datasets, i.e., FreeSolv, ESOL and QM9 [68]. FreeSolv and ESOL are small-scale molecular datasets. QM9 is a large-scale-molecular dataset. For each experiment, 20% of randomly selected graphs from the original dataset serves as the target dataset for training the target GNN classifier. Additionally, another 10% of the graphs are set as the validation set, while the remaining 70% are assigned as the test set. Note that we set the test set as the auxiliary datase for attackers to train the graph generator and conduct reconstruction. The statistics of the datasets are summarized in Tab. 1. More details of the dataset settings are shown in *Appendix B.1.*

5.1.2 Baselines. To the best of our knowledge, GraphSteal is the first graph stealing attack to extract training graphs without access to the private graph-level dataset. To demonstrate the effectiveness of GraphSteal, we first propose three variants:

- **BL-Rand**: It randomly generates graphs using Erdos-Renyi (ER) models, where node features are randomly sampled from the graphs in the auxiliary dataset.
- **BL-Conf**: This method directly selects the top-k most confidence graphs from the auxiliary dataset as the reconstructed graphs.
- **BL-Diff**: It applies the graph diffusion model trained on the auxiliary datasets to generate graphs without any modification and selection.

Moreover, we compare GraphSteal with a state-of-the-art model inversion attack method for graph neural networks:

- **GraphMI-G**: This method is extended from *GraphMI* [82], which is designed for node classification. Specifically, GraphMI aims to reconstruct the adjacency matrix of the target graphs based on the node features and labels of the target graphs. To adapt it to our setting, where there is no partial information about the target dataset, we randomly select graphs from the auxiliary dataset as input graphs. We then apply GraphMI to modify the adjacency matrices of these graphs. The modified graphs are finally regarded as the reconstructed graphs.

Additionally, we also consider a state-of-the-art model-level explanation method as the baseline:

- **XGNN** [76]: It generates representative graphs for each class that the target classifier is most confident with as model-level explanations. To adapt it to our setting, we treat these generated representative graphs as reconstructed training graphs.

Table 1: Statistics of Datasets

Datasets	#Graphs	#Avg. Nodes	#Avg. Edges	#Avg. Feature	#Classes
FreeSolv	642	8.7	16.8	9	2
ESOL	1,128	13.3	27.4	9	2
QM9	130,831	8.8	18.8	11	3

5.1.3 Implementation Details. In this paper, we conduct experiments on the inductive supervised graph classification task, *where the adversary aims to extract training graphs from the privately owned dataset.* To demonstrate the transferability of GraphSteal, we target GNNs with various architectures, i.e., 2-layer GCN [35], 2-layer GIN [70] and 9-layer Graph Transformer (GTN) [17]. DiGress [60] is set as the basic graph diffusion model to generate graphs. All hyperparameters of the compared methods are tuned for fair comparisons. Each experiment is conducted 5 times on an A6000 GPU with 48G memory and the average results are reported.

5.1.4 Evaluation Metrics. *First*, we adopt the following metrics to evaluate the *realism* and *quality* of the reconstructed graphs:

- **Validity** calculates the fraction of the reconstructed molecular graphs adhering to fundamental chemical rules and principles as Validity $= V(\mathcal{D}_r)/|\mathcal{D}_r|$, where $V(\mathcal{D}_r)$ is the number of graphs in $\mathcal{D}_r$ adhering to fundamental chemical rules and principles measured by RDKit [1]. A larger validity score implies the better quality and realism of the reconstructed graphs.

[1] https://www.rdkit.org/

Table 2: Comparison with baselines in stealing training graphs on various graph datasets against GCN.

Dataset	Metrics	BL-Rand	BL-Conf	BL-Diff	XGNN	GraphMI-G	GraphSteal
FreeSolv	Validity (%) ↑	20.3±5.4	**100±0**	99.3±0.8	17.0±2.5	33.0±1.2	98.8±0.8
	Uniqueness (%) ↑	97.4±3.6	95.6±0.3	77.0±3.2	**100±0**	66.7±0.5	81.8±1.8
	Recon. Rate (%) ↑	4.0±3.4	2.2±0.8	39.4±4.2	1.9±1.8	4.5±0.9	**50.4±2.3**
	FCD ↓	16.9±1.2	9.5±0.9	6.6±0.3	23.4±1.1	16.6±0.5	**5.7±0.5**
ESOL	Validity (%)↑	15.0±2.8	**100±0**	98.7±1.2	14.7±1.7	14.3±0.5	96.7±2.2
	Uniqueness (%)↑	**100±0**	95.0±0.4	94.5±0.9	100±0	86.0±0.5	95.6±0.8
	Recon. Rate (%) ↑	0±0	0±0	4.8±0.8	0±0	0±0	**8.6±1.4**
	FCD ↓	27.0±2.7	17.6±0.1	4.9±0.5	29.9±0.5	27.8±7.9	**4.5±0.3**
QM9	Validity (%) ↑	3.3±0.5	**100±0**	98.4±1.7	57.0±1.6	4.7±0.5	98.8±0.8
	Uniqueness (%) ↑	**100±0**	82.6±3.1	98.2±2.7	98.2±1.5	**100±0**	**100±0**
	Recon. Rate (%) ↑	0±0	1.9±0.9	19.4±2.8	0±0	0±0	**29.2±3.1**
	FCD ↓	21.6±2.3	9.6±0.8	2.8±1.8	12.2±0.5	19.9±1.1	**2.0±0.2**

- **Uniqueness** measures the proportion of graphs that exhibit uniqueness within the entire set of reconstructed graphs, which is calculated by Uniqueness = $U(\mathcal{D}_r)/|\mathcal{D}_r|$, where $U(\mathcal{D}_r)$ is the number of unique graphs within $\mathcal{D}_r$. A larger uniqueness score implies a better diversity in reconstruction.

Second, we apply the following metrics to evaluate the *fidelity* of the reconstructed graphs in mirroring the graphs in the target dataset:

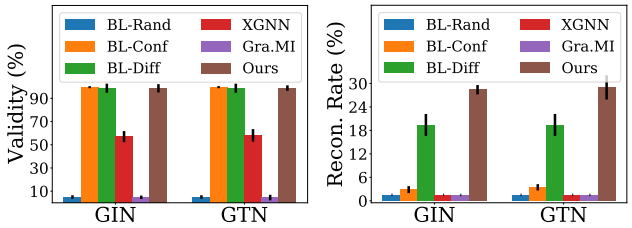
- **Reconstruction Rate** represents the fraction of reconstructed graphs matching those in the target dataset as Reconstruction rate = $|\mathcal{D}_r \cap \mathcal{D}_t|/|\mathcal{D}_r|$, where $|\mathcal{D}_r \cap \mathcal{D}_t|$ denotes the number of reconstructed graphs exactly matched with those in $\mathcal{D}_t$. A larger reconstruction rate indicates a better reconstruction method. More details of the exact graph matching are in Appendix B.2.2.
- **Fréchet ChemNet Distance (FCD)** [55] quantifies the distance between the distributions of the representations of the reconstructed dataset and the target dataset. The representations are learned from the pre-trained ChemNet [23] neural networks. More details of the computation process of FCD is in Appendix B.2.1. A lower FCD indicates a better reconstruction method.

5.2 Reconstruction Results

To answer **Q1**, we compare GraphSteal with baselines in reconstructing private graphs from the target training dataset on three molecular graph datasets. We also evaluate the performance of GraphSteal against various GNN models to validate its flexibility. Moreover, the impacts of the training/auxiliary set distribution shift and split ratio are further investigated in *Appendix D*.

5.2.1 Comparisons with Baselines. We focus on attacking the GNNs for the graph classification task. The target GNN is set as GCN with a sum pooling layer. The final selection number is set as 100. The average reconstruction results on FreeSolv, ESOL, and QM9 are reported in Tab. 2. Note that a lower FCD indicates a better reconstruction performance. From Tab. 2, we can observe that:

- Existing baselines give poor performance in validity, reconstruction rate and FCD among all datasets. It implies the necessity of developing graph stealing attacks for reconstructing realistic and high-quality graphs from the privately owned training dataset.
- GraphSteal gives superior validity and uniqueness than baselines. It indicates the effectiveness of GraphSteal in reconstructing realistic graphs that are valid in real-world scenarios even on the large-scale dataset QM9.
- GraphSteal shows a much better reconstruction rate and FCD than baselines. This demonstrates GraphSteal can obtain high-quality reconstructions by generating and selecting graphs that closely match those in the target dataset.



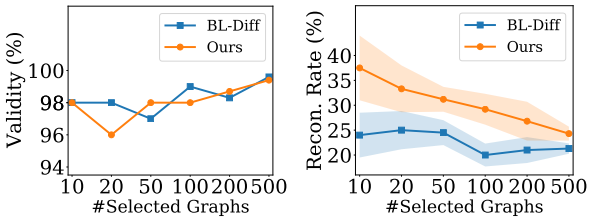
(a) Validity **(b) Recon. Rate**

Figure 3: Reconstruction results on QM9 for various GNNs.

5.2.2 Flexibility to Model Architecture. To show the flexibility of GraphSteal to different GNN models, we consider two more GNNs, i.e., GIN [70] and GTN [17], as the target models to conduct graph stealing attack. The number of selected reconstructed graphs is set as 100. All other settings are the same as that in Sec. 5.1.3. We compare GraphSteal with baselines on QM9. The average results of validity, and reconstruction rate are reported in Fig. 3. Similar trends are also observed on other datasets. From the figure, we observe that GraphSteal consistently achieves better reconstruction rate, while maintaining high validity, compared to baselines for both GIN and GTN models. It demonstrates the effectiveness of GraphSteal in reconstructing realistic and high-quality graphs for various GNN models, which shows the flexibility of GraphSteal to steal graphs from various GNN classifiers.

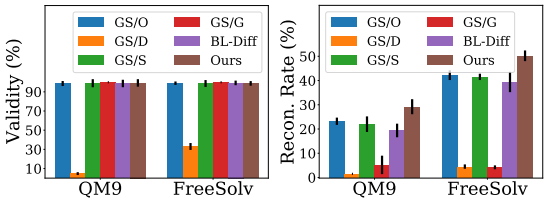
5.3 Impact of the Number of Selected Graphs

To answer **Q2**, we conduct experiments to explore the attack performance of GraphSteal given different budgets in the numbers of selected graphs after reconstruction. Specifically, we vary the number of selected graphs as {10, 20, 50, 100, 200, 500}. GCN is set as the target GNN model. The other settings are the same as Sec. 5.1.3. Fig. 4 reports the validity and reconstruction rate on QM9 datasets.



(a) Validity (b) Recon. Rate

Figure 4: Impact of the numbers of selected graphs on QM9



(a) Validity (b) Recon. Rate

Figure 5: Ablation studies on QM9 and FreeSolv.

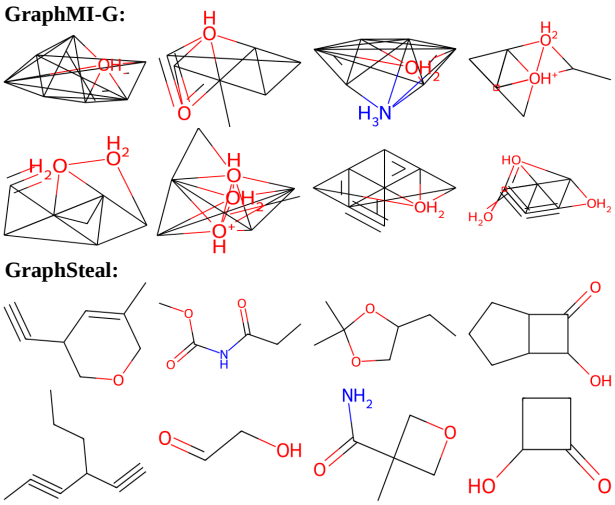


Figure 6: Reconstructed training graphs of QM9.

More results on other datasets can be found in *Appendix C*. From the figure, we observe that:

- As the number of selected graphs increases, the reconstruction rate of our method would slightly decrease. However, it is worth noting that the absolute number of the matched reconstructed graphs (# Selected Graphs × Recon. Rate) increases, which satisfies our expectation. This is because according to Theorem 4.1, the trained GNN classifier f_{θ} is regarded as a weighted combination of a subset of the derivative of training graphs. Thus, there is a theoretical upper bound to the number of graphs that can be reconstructed based on θ. Moreover, our proposed reconstruction selection method can effectively identify the reconstructed graphs that are more likely to be training graphs, leading to a high reconstruction rate when the selected graph number is small. As the number of selected graphs increases, the number of matched reconstructed graphs will gradually reach its upper bound, resulting in a diminished reconstruction rate. More details of the absolute number of matched reconstructed graphs are in *Appendix C*.
- Our method consistently outperforms BL-diff in terms of reconstruction rate as the number of selected graphs increases, while still maintaining high validity, which demonstrates the effectiveness of our method in reconstructing realistic and high-quality graphs in real-world scenarios.

5.4 Ablation Study

To answer **Q3**, we conduct ablation studies to understand the effects of the reconstruction generation and reconstruction selection method. (i) To demonstrate the effectiveness of the diffusion noise optimization in Sec.4.1.2, we implement a variant GraphSteal/O that replaces the optimized graphs with the Gaussian noise as the input to the graph diffusion model. (ii) GraphSteal deploys a graph diffusion model to improve the realism and quality of reconstruction. To prove its effectiveness, we directly regard the optimized graphs in Sec.4.1.2 as the reconstructed graphs and train a variant named GraphSteal/D. (iii) A variant named GraphSteal/S is trained by replacing the reconstruction selection component with the random selection. (iv) We also propose a variant named GraphSteal/G by directly conducting selection from the auxiliary dataset based on Eq. (15) without generating graphs to show the effectiveness of the overall reconstruction generation. BL-Diff is also adopted as a reference. GCN is set as the target GNN model. The number of selected graphs is set as 100.

The average results on FreeSolv and QM9 are reported in Fig. 5. From the figure, we observe that: **(i)** GraphSteal/O and GraphSteal/S perform significantly worse than GraphSteal in terms of reconstruction rate, while they still outperform BL-Diff. It shows the effectiveness of our proposed diffusion noise optimization and graph selection mask optimization, respectively; **(ii)** GraphSteal outperforms GraphSteal/D and GraphSteal/G by a large margin in reconstruction rate. This demonstrates the proposed graph generator can effectively enhance the realism and quality of the reconstructed graphs; and **(iii)** GraphSteal/D exhibits extremely lower validity than all other methods. This aligns with our description in Sec. 4.1.3 that directly using graphs optimized based on Sec. 4.1.2 as reconstructions would encounter issues of unrealism and invalidity.

5.5 Reconstruction Visualization

In this subsection, we conduct a case study to further demonstrate the effectiveness of GraphSteal. We conduct graph stealing attack using both GraphMI-G and GraphSteal on QM9 and then visualize the top-8 representative reconstructed graphs. The visualizations are plotted in Fig. 6. From the figure, we observe that the graphs reconstructed by GraphMI-G appear unrealistic and invalid, which implies the poor performance of GraphMI-G in graph stealing attacks. In contrast, the graphs reconstructed by our method exhibit both high quality and realism, thereby verifying the effectiveness of our method in reconstructing high-quality graphs by leveraging the model parameters of GNNs.

6 CONCLUSION AND FUTURE WORK

In this paper, we study a novel privacy attack problem of stealing training graphs from trained GNNs. We propose a novel framework, GraphSteal, to reconstruct training graphs by leveraging the parameters of GNNs. Specifically, we apply a graph diffusion model as the graph generator to reconstruct graphs realistically. We then implement a diffusion noise optimization to enhance the resemblance of the graphs generated by the graph generator to the target training data. Moreover, a model parameter-guided graph selection method is proposed to identify the training graphs from the generated graphs by leveraging the parameters of GNNs. Extensive experiments on different real-world datasets demonstrate the effectiveness of GraphSteal in reconstructing realistic and high-quality training graphs from trained GNNs. There are two directions that need further investigation. First, in this paper, we only focus on performing graph stealing attack in the white-box setting. Thus, it is also interesting to in investigate how to conduct the black-box graph stealing attack effectively. Second, it is also worthwhile to investigate how to defend against the graph stealing attack. The discussions of the potential countermeasures and ethical implications are in *Appendix E* and *Appendix F*, respectively.

ACKNOWLEDGMENTS

This material is based upon work supported by, or in part by, the Army Research Office (ARO) under grant number W911NF-21-1-0198, the Department of Homeland Security (DHS) CINA under grant number E205949D, and the Cisco Faculty Research Award. The findings in this paper do not necessarily reflect the view of the funding agencies.

REFERENCES

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 308–318.

[2] Yogesh Balaji, Seungjun Nah, Xun Huang, Arash Vahdat, Jiaming Song, Karsten Kreis, Miika Aittala, Timo Aila, Samuli Laine, Bryan Catanzaro, et al. 2022. ediffi: Text-to-image diffusion models with an ensemble of expert denoisers. *arXiv preprint arXiv:2211.01324* (2022).

[3] Ting Chen. 2023. On the importance of noise scheduling for diffusion models. *arXiv preprint arXiv:2301.10972* (2023).

[4] Enyan Dai and Jie Chen. 2022. Graph-augmented normalizing flows for anomaly detection of multiple time series. *arXiv preprint arXiv:2202.07857* (2022).

[5] Enyan Dai, Limeng Cui, Zhengyang Wang, Xianfeng Tang, Yinghan Wang, Monica Cheng, Bing Yin, and Suhang Wang. 2023. A unified framework of graph information bottleneck for robustness and membership privacy. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 368–379.

[6] Enyan Dai, Minhua Lin, and Suhang Wang. 2024. PreGIP: Watermarking the Pretraining of Graph Neural Networks for Deep Intellectual Property Protection. *arXiv preprint arXiv:2402.04435* (2024).

[7] Enyan Dai, Minhua Lin, Xiang Zhang, and Suhang Wang. 2023. Unnoticeable backdoor attacks on graph neural networks. In *Proceedings of the ACM Web Conference 2023*. 2263–2273.

[8] Enyan Dai and Suhang Wang. 2021. Say no to the discrimination: Learning fair graph neural networks with limited sensitive attribute information. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. 680–688.

[9] Enyan Dai and Suhang Wang. 2021. Towards self-explainable graph neural network. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 302–311.

[10] Enyan Dai and Suhang Wang. 2022. Learning fair graph neural networks with limited and private sensitive attribute information. *IEEE Transactions on Knowledge and Data Engineering* 35, 7 (2022), 7103–7117.

[11] Enyan Dai and Suhang Wang. 2022. Towards prototype-based self-explainable graph neural network. *ACM Transactions on Knowledge Discovery from Data* (2022).

[12] Enyan Dai, Tianxiang Zhao, Huaisheng Zhu, Junjie Xu, Zhimeng Guo, Hui Liu, Jiliang Tang, and Suhang Wang. 2022. A comprehensive survey on trustworthy graph neural networks: Privacy, robustness, fairness, and explainability. *arXiv preprint arXiv:2204.08570* (2022).

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[14] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).

[15] Simon Shaolei Du, Wei Hu, Sham M Kakade, Jason D Lee, and Qi Lei. 2020. Few-Shot Learning via Learning the Representation, Provably. In *ICLR*.

[16] Vasisht Duddu, Antoine Boutet, and Virat Shejwalkar. 2020. Quantifying privacy leakage in graph embedding. In *MobiQuitous 2020-17th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*. 76–85.

[17] Vijay Prakash Dwivedi and Xavier Bresson. 2021. A Generalization of Transformer Networks to Graphs. *AAAI Workshop on Deep Learning on Graphs: Methods and Applications* (2021).

[18] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. 2019. A Fair Comparison of Graph Neural Networks for Graph Classification. In *ICLR*.

[19] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph neural networks for social recommendation. In *The world wide web conference*. 417–426.

[20] Xiang Fang, Arvind Easwaran, Blaise Genest, and Ponnuthurai Nagaratnam Suganthan. 2025. Your data is not perfect: Towards cross-domain out-of-distribution detection in class-imbalanced data. *Expert Systems with Applications* 267 (2025), 126031.

[21] Xiang Fang, Wanlong Fang, Daizong Liu, Xiaoye Qu, Jianfeng Dong, Pan Zhou, Renfu Li, Zichuan Xu, Lixing Chen, Panpan Zheng, et al. 2024. Not all inputs are valid: Towards open-set video moment retrieval using language. In *ACM MM*.

[22] Xiang Fang, Daizong Liu, Wanlong Fang, Pan Zhou, Yu Cheng, Keke Tang, and Kai Zou. 2023. Annotations Are Not All You Need: A Cross-modal Knowledge Transfer Network for Unsupervised Temporal Sentence Grounding. In *Findings of EMNLP*.

[23] Elyas Goli, Sagar Vyas, Seid Koric, Nahil Sobh, and Philippe H Geubelle. 2020. Chemnet: A deep neural network for advanced composites manufacturing. *The Journal of Physical Chemistry B* 124, 42 (2020), 9428–9437.

[24] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *NeurIPS* 30 (2017).

[25] Xinlei He, Jinyuan Jia, Michael Backes, Neil Zhenqiang Gong, and Yang Zhang. 2021. Stealing links from graph neural networks. In *30th USENIX Security Symposium (USENIX Security 21)*. 2669–2686.

[26] Xinlei He, Rui Wen, Yixin Wu, Michael Backes, Yun Shen, and Yang Zhang. 2021. Node-level membership inference attacks against graph neural networks. *arXiv preprint arXiv:2102.05429* (2021).

[27] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. 2017. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *NeurIPS* 30 (2017).

[28] Jonathan Ho, William Chan, Chitwan Saharia, Jay Whang, Ruiqi Gao, Alexey Gritsenko, Diederik P Kingma, Ben Poole, Mohammad Norouzi, David J Fleet, et al. 2022. Imagen video: High definition video generation with diffusion models. *arXiv preprint arXiv:2210.02303* (2022).

[29] Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising diffusion probabilistic models. *NeurIPS* 33 (2020), 6840–6851.

[30] Emiel Hoogeboom, Vıctor Garcia Satorras, Clément Vignac, and Max Welling. 2022. Equivariant diffusion for molecule generation in 3d. In *International conference on machine learning*. PMLR, 8867–8887.

[31] Zhichao Hou, Minhua Lin, MohamadAli Torkamani, Suhang Wang, and Xiaorui Liu. 2024. Adversarial Robustness in Graph Neural Networks: Recent Advances and New Frontier. In *2024 IEEE 11th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 1–2.

[32] Adrián Javaloy, Pablo Sanchez Martin, Amit Levi, and Isabel Valera. 2023. Learnable Graph Convolutional Attention Networks. In *International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=WsUMeHPo-2

[33] Ziwei Ji and Matus Telgarsky. 2020. Directional convergence and alignment in deep learning. *NeurIPS* 33 (2020), 17176–17186.

[34] Jaehyeong Jo, Seul Lee, and Sung Ju Hwang. 2022. Score-based generative modeling of graphs via the system of stochastic differential equations. In *International Conference on Machine Learning*. PMLR, 10362–10383.

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

[36] Greg Landrum. 2013. Rdkit documentation. *Release* 1, 1-79 (2013), 4.

[37] O-Joun Lee et al. 2024. Transitivity-Preserving Graph Representation Learning for Bridging Local Connectivity and Role-Based Similarity. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 12456–12465.

[38] Xiaoxiao Li, Yuan Zhou, Nicha Dvornek, Muhan Zhang, Siyuan Gao, Juntang Zhuang, Dustin Scheinost, Lawrence H Staib, Pamela Ventola, and James S Duncan. 2021. Braingnn: Interpretable brain graph neural network for fmri analysis. *Medical Image Analysis* 74 (2021), 102233.

[39] Yang Li, Buyue Qian, Xianli Zhang, and Hui Liu. 2020. Graph neural network-based diagnosis prediction. *Big Data* 8, 5 (2020), 379–390.

[40] Ke Liang, Lingyuan Meng, Hao Li, Meng Liu, Siwei Wang, Sihang Zhou, Xinwang Liu, and Kunlun He. 2024. MGKsite: Multi-Modal Knowledge-Driven Site Selection via Intra and Inter-Modal Graph Fusion. *IEEE Transactions on Multimedia* (2024).

[41] Ke Liang, Lingyuan Meng, Meng Liu, Yue Liu, Wenxuan Tu, Siwei Wang, Sihang Zhou, Xinwang Liu, Fuchun Sun, and Kunlun He. 2024. A survey of knowledge graph reasoning on graph types: Static, dynamic, and multi-modal. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2024).

[42] Ke Liang, Lingyuan Meng, Yue Liu, Meng Liu, Wei Wei, Suyuan Liu, Wenxuan Tu, Siwei Wang, Sihang Zhou, and Xinwang Liu. 2024. Simple Yet Effective: Structure Guided Pre-trained Transformer for Multi-modal Knowledge Graph Reasoning. In *Proceedings of the 32nd ACM International Conference on Multimedia*. 1554–1563.

[43] Ke Liang, Sihang Zhou, Meng Liu, Yue Liu, Wenxuan Tu, Yi Zhang, Liming Fang, Zhe Liu, and Xinwang Liu. 2024. Hawkes-enhanced spatial-temporal hypergraph contrastive learning based on criminal correlations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 8733–8741.

[44] Minhua Lin, Zhengzhang Chen, Yanchi Liu, Xujiang Zhao, Zongyu Wu, Junxiang Wang, Xiang Zhang, Suhang Wang, and Haifeng Chen. 2024. Decoding Time Series with LLMs: A Multi-Agent Framework for Cross-Domain Annotation. *arXiv preprint arXiv:2410.17462* (2024).

[45] Minhua Lin, Teng Xiao, Enyan Dai, Xiang Zhang, and Suhang Wang. 2024. Certifiably robust graph contrastive learning. *Advances in Neural Information Processing Systems* 36 (2024).

[46] Minhua Lin, Zhiwei Zhang, Enyan Dai, Zongyu Wu, Yilong Wang, Xiang Zhang, and Suhang Wang. 2024. Trojan Prompt Attacks on Graph Neural Networks. *arXiv preprint arXiv:2410.13974* (2024).

[47] Yi Liu, Limei Wang, Meng Liu, Yuchao Lin, Xuan Zhang, Bora Oztekin, and Shuiwang Ji. 2022. Spherical message passing for 3d molecular graphs. In *ICLR*.

[48] Tianze Luo, Zhanfeng Mo, and Sinno Jialin Pan. 2023. Fast graph generation via spectral diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2023).

[49] Kaifeng Lyu and Jian Li. 2020. Gradient Descent Maximizes the Margin of Homogeneous Neural Networks. In *ICLR*.

[50] Chenlin Meng, Yutong He, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, and Stefano Ermon. 2022. SDEdit: Guided Image Synthesis and Editing with Stochastic Differential Equations. In *ICLR*.

[51] Luis Müller, Mikhail Galkin, Christopher Morris, and Ladislav Rampášek. 2023. Attending to graph transformers. *arXiv preprint arXiv:2302.04181* (2023).

[52] Mor Shpigel Nacson, Suriya Gunasekar, Jason Lee, Nathan Srebro, and Daniel Soudry. [n. d.]. Lexicographic and depth-sensitive margins in homogeneous and non-homogeneous deep models. In *ICML*.

[53] Iyiola E Olatunji, Wolfgang Nejdl, and Megha Khosla. 2021. Membership inference attack on graph neural networks. In *2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*. IEEE, 11–20.

[54] Noel M O'Boyle and Roger A Sayle. 2016. Comparing structural fingerprints using a literature-based similarity benchmark. *Journal of cheminformatics* 8 (2016), 1–14.

[55] Kristina Preuer, Philipp Renz, Thomas Unterthiner, Sepp Hochreiter, and Günter Klambauer. 2018. Fréchet ChemNet Distance: A Metric for Generative Models for Molecules in Drug Discovery. *Journal of Chemical Information and Modeling* 58, 9 (2018), 1736–1741.

[56] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. 2022. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125* 1, 2 (2022), 3.

[57] Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. 2022. Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems* 35 (2022), 14501–14515.

[58] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. 2021. Score-Based Generative Modeling through Stochastic Differential Equations. In *ICLR*. https://openreview.net/forum?id=PxTIG12RRHS

[59] Leonid Nisonovich Vaserstein. 1969. Markov processes over denumerable products of spaces, describing large systems of automata. *Problemy Peredachi Informatsii* 5, 3 (1969), 64–72.

[60] Clement Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. 2023. DiGress: Discrete Denoising diffusion for graph generation. In *ICLR*.

[61] Binghui Wang, Minhua Lin, Tianxiang Zhou, Pan Zhou, Ang Li, Meng Pang, Hai Li, and Yiran Chen. 2024. Efficient, direct, and restricted black-box graph evasion attacks to any-layer graph neural networks via influence function. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*. 693–701.

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

[63] Yuyang Wang, Jianren Wang, Zhonglin Cao, and Amir Barati Farimani. 2022. Molecular contrastive learning of representations via graph neural networks. *Nature Machine Intelligence* 4, 3 (2022), 279–287.

[64] David Weininger. 1988. SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. *Journal of chemical information and computer sciences* 28, 1 (1988), 31–36.

[65] Bang Wu, Xiangwen Yang, Shirui Pan, and Xingliang Yuan. 2022. Model extraction attacks on graph neural networks: Taxonomy and realisation. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*. 337–350.

[66] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *International conference on machine learning*. 6861–6871.

[67] Yixin Wu, Xinlei He, Pascal Berrang, Mathias Humbert, Michael Backes, Neil Zhenqiang Gong, and Yang Zhang. 2024. Link Stealing Attacks Against Inductive Graph Neural Networks. *arXiv preprint arXiv:2405.05784* (2024).

[68] Zhenqin Wu, Bharath Ramsundar, Evan N Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S Pappu, Karl Leswing, and Vijay Pande. 2018. MoleculeNet: a benchmark for molecular machine learning. *Chemical science* 9, 2 (2018), 513–530.

[69] Junjie Xu, Zongyu Wu, Minhua Lin, Xiang Zhang, and Suhang Wang. 2024. LLM and GNN are Complementary: Distilling LLM for Multimodal Graph Learning. *arXiv preprint arXiv:2406.01032* (2024).

[70] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How Powerful are Graph Neural Networks?. In *ICLR*.

[71] Minkai Xu, Meng Liu, Wengong Jin, Shuiwang Ji, Jure Leskovec, and Stefano Ermon. 2023. Graph and geometry generative modeling for drug discovery. In *SIGKDD*. 5833–5834.

[72] Minkai Xu, Lantao Yu, Yang Song, Chence Shi, Stefano Ermon, and Jian Tang. 2022. Geodiff: A geometric diffusion model for molecular conformation generation. *arXiv preprint arXiv:2203.02923* (2022).

[73] Haotian Xue, Alexandre Araujo, Bin Hu, and Yongxin Chen. 2023. Diffusion-Based Adversarial Sample Generation for Improved Stealthiness and Controllability. In *Thirty-seventh Conference on Neural Information Processing Systems*.

[74] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. 2021. Do transformers really perform badly for graph representation? *NeurIPS* 34 (2021), 28877–28888.

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

[76] Hao Yuan, Jiliang Tang, Xia Hu, and Shuiwang Ji. 2020. Xgnn: Towards model-level explanations of graph neural networks. In *SIGKDD*. 430–438.

[77] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. 2021. Understanding deep learning (still) requires rethinking generalization. *Commun. ACM* 64, 3 (2021), 107–115.

[78] Jiawei Zhang, Haopeng Zhang, Congying Xia, and Li Sun. 2020. Graph-bert: Only attention is needed for learning graph representations. *arXiv preprint arXiv:2001.05140* (2020).

[79] Zhikun Zhang, Min Chen, Michael Backes, Yun Shen, and Yang Zhang. 2022. Inference attacks against graph neural networks. In *31st USENIX Security Symposium (USENIX Security 22)*. 4543–4560.

[80] Zhiwei Zhang, Minhua Lin, Enyan Dai, and Suhang Wang. 2024. Rethinking graph backdoor attacks: A distribution-preserving perspective. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 4386–4397.

[81] Zhiwei Zhang, Minhua Lin, Junjie Xu, Zongyu Wu, Enyan Dai, and Suhang Wang. 2024. Robustness-Inspired Defense Against Backdoor Attacks on Graph Neural Networks. *arXiv preprint arXiv:2406.09836* (2024).

[82] Zaixi Zhang, Qi Liu, Zhenya Huang, Hao Wang, Chengqiang Lu, Chuanren Liu, and Enhong Chen. 2021. GraphMI: Extracting Private Graph Data from Graph Neural Networks. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*. 3749–3755.

A DETAILED PROOFS

A.1 Preliminaries of KKT Conditions

In this subsection, we present the backgrounds of the Karush-Kuhn-Tucker (KKT) condition.

Consider the following optimization problem:

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}) \\ &\text{s.t.} \quad \delta_n(\mathbf{x}) \le 0 \quad \forall n \in [N], \end{aligned} \tag{18}$$

where $f, \delta_1, \ldots, \delta_n : \mathbb{R}^d \to \mathbb{R}$ are locally Lipschitz functions. We say that $\mathbf{x} \in \mathbb{R}^d$ is a feasible point of Eq. (18) if $\mathbf{x}$ satisfies $\delta_n(\mathbf{x}) \le 0$ for all $n \in [N]$. This constrained optimization problem can then form the Lagrangian function:

$$L = f(\mathbf{x}) + \sum_{n=1}^{N} \lambda_n \delta_n(\mathbf{x}), \tag{19}$$

if there exist $\lambda_n \in \lambda$ satisfies the following conditions:

$$\nabla_{\mathbf{x}} L = \nabla f + \sum_{n=1}^{N} \nabla \delta_n = \mathbf{0} \tag{20}$$

$$\delta_n(\mathbf{x}) \le 0, \tag{21}$$

$$\lambda_1, \ldots, \lambda_N \ge 0, \tag{22}$$

$$\lambda_n \delta_n(\mathbf{x}) = 0, \quad \forall i \in [N], \tag{23}$$

where Eq. (20) to Eq. (23) are the stationarity, primal feasibility, dual feasibility and complementary slackness of KKT conditions, respectively. Then we give the definition of KKT point as follows:

Definition A.1 (KKT point). *A feasible point* $\mathbf{x}$ *of the optimization problem in Eq. (18) is KKT point if* $\mathbf{x}$ *satisfies KKT conditions in Eq. (20) to Eq. (23).*

Note that a global minimum of Eq. (18) may not be a KKT point, but under some regularity assumptions (e.g., Mangasarian-Fromovitz Constraint Qualification), the KKT conditions will then be the necessary condition for global optimality.

A.2 Homogeneity of Graph Neural Networks

Before we start to discuss the homogeneity of GNNs, we recall the definition of homogeneous neural networks in Sec. 4.2.1 as follows:

Definition 4.1 (Homogeneous Neural Networks [49]). *Let* f *be a neural network with model parameters as* $\boldsymbol{\theta}$*. Then* f *is a homogeneous neural network if there is a number* $L > 0$ *such that the model output* $f(\mathcal{G}; \boldsymbol{\theta})$ *satisfies the following equation:*

$$f(\mathcal{G}; \sigma\boldsymbol{\theta}) = \sigma^L f(\mathcal{G}; \boldsymbol{\theta}), \ \forall \sigma > 0. \tag{24}$$

Following [49, 52], any fully-connected or convolutional neural network with ReLU activations is homogeneous w.r.t the mode parameters $\boldsymbol{\theta}$ if it does not have any bias terms or skip-connections. To extend to GNNs, the message passing in GNNs can be regarded as a generalized form of convolution from the spectral perspective [35]. Therefore, we can claim that any message-passing based GNNs (e.g. GCN [35] and SGC [66]) with ReLU activations is homogeneous w.r.t the parameters $\boldsymbol{\theta}$ if it does not have any skip-connections (e.g., GraphSage [24]) or bias terms, except possibly for the first layer. Our experimental results in Sec. 5.2.2 also demonstrate the effectiveness of GraphSteal across various GNN architectures.

We further take SGC [66] and GCN as examples.

A.2.1 Proof for SGC. Given a K-layer SGC, the model output $f(\mathcal{G}; \boldsymbol{\theta})$ is:

$$f(\mathcal{G}; \boldsymbol{\theta}) = \mathbf{S}^K \mathbf{X} \boldsymbol{\theta}, \tag{25}$$

where $\mathbf{S} = \tilde{\mathbf{D}}^{-1/2} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-1/2}$, $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ and $\tilde{\mathbf{D}}$ is the degree matrix of $\tilde{\mathbf{A}}$. $\boldsymbol{\theta} = \boldsymbol{\theta}^{(1)} \boldsymbol{\theta}^{(2)} \ldots \boldsymbol{\theta}^{(L)}$ Therefore, for $f(\mathcal{G}; \sigma\boldsymbol{\theta})$, we have:

$$\begin{aligned} f(\mathcal{G}; \sigma\boldsymbol{\theta}) &= \sigma \mathbf{S}^K \mathbf{X} \boldsymbol{\theta} \\ &= \sigma f(\mathcal{G}; \boldsymbol{\theta}). \end{aligned} \tag{26}$$

Thus, $f(\mathcal{G}; \boldsymbol{\theta})$ satisfies Eq. (24) when $L = 1$, which shows that SGC is a homogeneous GNN.

A.2.2 Proof for GCN. GCN [35] can also be proved as a homogeneous GNN in a similar way as GCN is the ReLU version of SGC. We first prove the homogeneity of a single layer of GCN. Specifically, for the i-th layer of GCN, the output is:

$$f^{(i)}(\mathbf{H}^{(i-1)}; \boldsymbol{\theta}^{(i)}) = \mathrm{ReLU}(\mathbf{S}\mathbf{H}^{(i-1)} \boldsymbol{\theta}^{(i)}), \quad i \in \{1, \ldots, K\} \tag{27}$$

where $\mathbf{H}^{(i-1)} = f^{(i-1)}(\mathbf{H}^{(i-2)}; \boldsymbol{\theta}^{(i-1)})$ is the output from the $(i-1)$-th layer of a K-layer GCN, and $\mathbf{H}^{(0)} = \mathbf{X}$.

Then, we apply the positive scalar $\sigma > 0$ to $\boldsymbol{\theta}^{(i)}$ and obtain

$$\begin{aligned} f^{(i)}(\mathbf{H}^{(i-1)}; \sigma\boldsymbol{\theta}^{(i)}) &= \mathrm{ReLU}(\mathbf{S}\mathbf{H}^{(i-1)} \cdot \sigma\boldsymbol{\theta}^{(i)}) \\ &= \mathrm{ReLU}(\sigma \cdot \mathbf{S}\mathbf{H}^{(i-1)} \boldsymbol{\theta}^{(i)}). \end{aligned} \tag{28}$$

If $\mathbf{S}\mathbf{H}^{(i-1)} \boldsymbol{\theta}^{(i)} > 0$, we have

$$\begin{aligned} \mathrm{ReLU}(\sigma \cdot \mathbf{S}\mathbf{H}^{(i-1)} \boldsymbol{\theta}^{(i)}) &= \sigma \cdot \mathbf{S}\mathbf{H}^{(i-1)} \boldsymbol{\theta}^{(i)} \\ &= \sigma \cdot \mathrm{ReLU}(\mathbf{S}\mathbf{H}^{(i-1)} \boldsymbol{\theta}^{(i)}). \end{aligned} \tag{29}$$

And if $\mathbf{S}\mathbf{H}^{(i-1)} \boldsymbol{\theta}^{(i)} \le 0$, we have

$$\mathrm{ReLU}(\sigma \cdot \mathbf{S}\mathbf{H}^{(i-1)} \boldsymbol{\theta}^{(i)}) = 0 \tag{30}$$

for all $\sigma > 0$, which is consistent with $\mathrm{ReLU}(\mathbf{S}\mathbf{H}^{(i-1)} \boldsymbol{\theta}^{(i)}) = \sigma \cdot \mathrm{ReLU}(\mathbf{S}\mathbf{H}^{(i-1)} \boldsymbol{\theta}^{(i)}) = 0$.

Combining these two cases, for the i-th GCN layer $f^{(i)}(\mathbf{H}^{(i-1)}; \boldsymbol{\theta}^{(i)}) = \mathrm{ReLU}(\mathbf{S}\mathbf{H}^{(i-1)} \boldsymbol{\theta}^{(i)})$, we have

$$f^{(i)}(\mathbf{H}^{(i-1)}; \sigma\boldsymbol{\theta}^{(i)}) = \sigma \cdot f^{(i)}(\mathbf{H}^{(i-1)}; \sigma\boldsymbol{\theta}^{(i)}). \tag{31}$$

Therefore, there exist $L = 1$ such that $f^{(i)}(\mathbf{H}^{(i-1)}; \sigma\boldsymbol{\theta}^{(i)}) = \sigma^L f^{(i)}(\mathbf{H}^{(i-1)}; \sigma\boldsymbol{\theta}^{(i)})$. Hence, $f^{(i)}(\mathcal{G}; \sigma\boldsymbol{\theta}^{(i)})$ is a homogeneous function.

With this, we extend the proof to multiple layers of GCN. Since the output of one layer becomes the input to the next layer, based on Eq. (27), for the $(K-1)$-th GCN layer $f^{(K-1)}(\mathcal{G}; \theta)$, we have

$$\begin{aligned} f^{(K)}(\mathcal{G}; \boldsymbol{\theta}^{(K)}) &= \mathrm{ReLU}(\mathbf{S} f^{(K-1)}(\mathcal{G}; \boldsymbol{\theta}^{(K-1)}) \boldsymbol{\theta}^{(K)}) \\ &= \mathrm{ReLU}(\mathbf{S}\mathbf{H}^{(K-1)} \boldsymbol{\theta}^{(K)}). \end{aligned} \tag{32}$$

When the parameter $\theta^{(1)}$ of the first layer GCN is scaled by σ, we have

$$\begin{aligned} f^{(1)}(\mathcal{G}; \sigma\boldsymbol{\theta}^{(1)}) &= \mathrm{ReLU}(\mathbf{S}\mathbf{H}^{(i-1)} \cdot \sigma\boldsymbol{\theta}^{(i)}) \\ &= \sigma f^{(1)}(\mathcal{G}; \boldsymbol{\theta}^{(1)}) \\ &= \sigma \mathbf{H}^{(1)} \end{aligned} \tag{33}$$

Table 3: Averaged results of the absolute number of the reconstructed graphs.

Dataset	10	20	50	100	200	500
FreeSolv	5.9	11.4	27.5	50.4	92.6	216.0
QM9	3.8	6.7	15.6	29.2	100	121.5

Then, when input Eq. (33) to the second layer GCN, we have

$$\begin{aligned} f^{(2)}(\sigma\mathbf{H}^{(1)};\boldsymbol{\theta}^{(2)}) &= \text{ReLU}(\mathbf{S}\cdot\sigma\mathbf{H}^{(1)}\boldsymbol{\theta}^{(2)}) \\ &= \text{ReLU}(\mathbf{S}\mathbf{H}^{(1)}\cdot\sigma\boldsymbol{\theta}^{(2)}) \\ &= f^{(2)}(\mathbf{H}^{(1)};\sigma\boldsymbol{\theta}^{(2)}) \\ &= \sigma\cdot\text{ReLU}(\mathbf{S}\mathbf{H}^{(1)}\boldsymbol{\theta}^{(2)}) \\ &= \sigma f^{(2)}(\mathbf{H}^{(1)};\boldsymbol{\theta}^{(2)}) \\ &= \sigma\mathbf{H}^{(2)}. \end{aligned} \tag{34}$$

Similarly, we can easily conclude that the output of the $(K-1)$-th layer is

$$f^{(K-1)}(\sigma\mathbf{H}^{(\mathbf{K-2})};\boldsymbol{\theta}^{(K-1)}) = \sigma\mathbf{H}^{(K-1)}. \tag{35}$$

By using the homogeneity of the single-layer GCN, we can have

$$\begin{aligned} f^{(K)}(\mathbf{H}^{(\mathbf{K-1})};\sigma\boldsymbol{\theta}^{(K)}) &= \text{ReLU}(\mathbf{S}\cdot\sigma\mathbf{H}^{(K-1)}\boldsymbol{\theta}^{(K)}) \\ &= \text{ReLU}(\mathbf{S}\mathbf{H}^{(K-1)}\cdot\sigma\boldsymbol{\theta}^{(K)}) \\ &= \sigma\cdot\text{ReLU}(\mathbf{S}\mathbf{H}^{(K-1)}\boldsymbol{\theta}^{(K)}) \\ &= \sigma\cdot f^{(K)}(\mathbf{H}^{(\mathbf{K-1})};\boldsymbol{\theta}^{(K)}). \end{aligned} \tag{36}$$

Hence, we can conclude that for the K layer GCN, $f(\mathcal{G};\sigma\boldsymbol{\theta}) = \sigma\cdot f(\mathcal{G};\boldsymbol{\theta})$, the homogeneity of the K layer GCN is then proved.

Note that our theorem is also applicable to LeakyReLU activation function. The proof is similar to the above.

A.3 Proof of Theorem 4.1

Before beginning our proof, we first recall Lemma 4.1 in Sec. 4.2.1.

Lemma 4.1 ([49]). *Let $f_{\boldsymbol{\theta}}$ be a homogeneous ReLU neural network with model parameters $\boldsymbol{\theta}$. Let $\mathcal{D}_t = \{(\mathcal{G}_i, y_i)\}_{i=1}^{|\mathcal{D}_t|}$ be the classification training dataset of $f_{\boldsymbol{\theta}}$, where $y_i \in \{1,\dots,C\}$. Assume $f_{\boldsymbol{\theta}}$ is trained by minimizing the cross entropy loss $\mathcal{L}_{ce} = \sum_{i=1}^{|\mathcal{D}_t|}\log(1+\sum_{j\neq y_i} e^{-q_{ij}(\boldsymbol{\theta})})$ over $\mathcal{D}_t$ using gradient flow, where $q_{ij}(\boldsymbol{\theta}) = f_{\boldsymbol{\theta}}(\mathcal{G}_i)_{y_i} - f_{\boldsymbol{\theta}}(\mathcal{G}_i)_j$ and $f_{\boldsymbol{\theta}}(\mathcal{G}_i)_{y_i}$ denotes the y_i-th entry of the logit score vector $f_{\boldsymbol{\theta}}(\mathcal{G}_i)$ before applying the softmax normalization. Then, gradient flow converges in direction towards a first-order stationary point of the following max-margin optimization problem:*

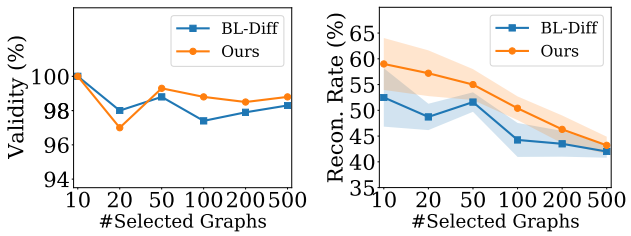
$$\min_{\boldsymbol{\theta}} \frac{1}{2}||\boldsymbol{\theta}||_2^2, \quad s.t. \quad q_{ij}(\boldsymbol{\theta}) \geq 1,\ \forall i \in [|\mathcal{D}_t|], j \in [C]\backslash\{y_i\}. \tag{37}$$

Given a homogeneous ReLU GNN model $f_{\boldsymbol{\theta}}$, according to Lemma 4.1, the training of $f_{\boldsymbol{\theta}}$ using gradient descent can be viewed as solving a max-margin optimization problem to maximize the margin $q_{ij}(\boldsymbol{\theta}) = f_{\boldsymbol{\theta}}(\mathcal{G}_i)_{y_i} - f_{\boldsymbol{\theta}}(\mathcal{G}_i)_j$ over all possible directions.

Moreover, the optimization problem in Eq. (37) can be reformulated to the following constrained optimization problem:

$$\min_{\boldsymbol{\theta}} \frac{1}{2}||\boldsymbol{\theta}||_2^2, \quad \text{s.t.} \quad \beta_i(\boldsymbol{\theta}) \geq 1,\ \forall i \in [|\mathcal{D}_t|], \tag{38}$$

where $\beta_i(\boldsymbol{\theta}) = f_{\boldsymbol{\theta}}(\mathcal{G}_i)_{y_i} - \max_{j\neq y_i}\{f_{\boldsymbol{\theta}}(\mathcal{G}_i)_j\}$ indicates the margin of $f_{\boldsymbol{\theta}}$ for a single data point $(\mathcal{G}_i, y_i)$. Following from [49], it can be proved that Eq. (38) satisfies the Mangasarian-Fromovitz Constraint Qualification (MFCQ). KKT conditions are then first-order necessary conditions for the global optimality of Eq. (38). Thus, after convergence of training $f_{\boldsymbol{\theta}}$ in direction to a KKT point $\tilde{\boldsymbol{\theta}}$ using gradient descent, there exist $\lambda_1,\dots,\lambda_{|\mathcal{D}_t|} \in \mathbb{R}$ such that:



(a) Validity **(b) Recon. Rate**

Figure 7: Impact of the numbers of selected graphs on FreeSolv

$$\tilde{\boldsymbol{\theta}} = \sum_{i=1}^{|\mathcal{D}_t|} \lambda_i(\nabla_{\boldsymbol{\theta}} f_{\tilde{\boldsymbol{\theta}}}(\mathcal{G}_i)_{y_i} - \nabla_{\boldsymbol{\theta}} \max_{j\neq y_i}\{f_{\tilde{\boldsymbol{\theta}}}(\mathcal{G}_i)_j\}), \tag{39}$$

$$\beta_i(\tilde{\boldsymbol{\theta}}) = f_{\tilde{\boldsymbol{\theta}}}(\mathcal{G}_i)_{y_i} - \max_{j\neq y_i}\{f_{\tilde{\boldsymbol{\theta}}}(\mathcal{G}_i)_j\} \geq 1,\ \forall i \in [|\mathcal{D}_t|] \tag{40}$$

$$\lambda_1,\dots,\lambda_{|\mathcal{D}_t|} \geq 0, \tag{41}$$

$$\lambda_i = 0 \quad \text{if } \beta_i(\tilde{\boldsymbol{\theta}}) \neq 1, \forall i \in |\mathcal{D}_t|, \tag{42}$$

where $\mathcal{G}_i \in \mathcal{D}_t$ denote the graph with ground-truth label y_i for training $f_{\boldsymbol{\theta}}$. Eq. (39) to Eq. (42) are the stationarity, primal feasibility, dual feasibility and complementary slackness of KKT conditions, respectively. Therefore, the KKT point $\tilde{\boldsymbol{\theta}}$ is the point that satisfy Eq. (11) in Theorem 4.1. This completes our proof.

B ADDITONAL DETAILS OF EXPERIMENT SETTINGS

B.1 Dataset Settings

We conduct experiments on 3 well-known datasets, i.e., FreeSolv, ESOL and QM9 [68], that are widely used for various graph-level tasks, including graph generation and graph regression. Given our focus on graph classification, we adapt these datasets to our setting by categorizing each graph according to the magnitude of its regression values and ensuring an equal division of graphs within each class. More specifically, for FreeSolv and ESOL, we partition the datasets into two classes, maintaining an equal number of graphs in each. To further validate the performance of GraphSteal in the multi-class graph classification task, we divide the QM9 dataset into three classes, also ensuring each class contains an equal number of graphs.

B.2 Additional Details of Evaluation Metrics

B.2.1 Fréchet ChemNet Distance (FCD). FCD [55] is a popular metric used to calculate the distance between the distribution $p_w(\cdot)$ of real-world molecules and the distribution $p(\cdot)$ of molecules from

a generative model. To calculate this distance, firstly, we use the activations of the penultimate layer of ChemNet [23] to obtain the representation of each molecule. We then calculate the mean and covariance of these activations for the two distributions. The two distributions $(p(\cdot), p_w(\cdot))$ are compared using the Wasserstein-2 distance [59]. Formally, FCD $d(\cdot,\cdot)$ is given by:

$$d^2((\mathbf{m},\mathbf{C}),(\mathbf{m}_w,\mathbf{C}_w)) = ||\mathbf{m}-\mathbf{m}_w||_2^2 + \mathrm{Tr}(\mathbf{C}+\mathbf{C}_w-2\cdot(\mathbf{C}\mathbf{C}_w)^{1/2}), \tag{43}$$

where $(\mathbf{m}_w, \mathbf{C}_w)$ are the mean and covariance of Gaussian $p_w(\cdot)$ obtained from real-world, and $(\mathbf{m}, \mathbf{C})$ are the mean and covariance of Gaussian $p(\cdot)$ obtained from the generative model. Note that FCD here is reported as $d^2(\cdot,\cdot)$ analogously to [27].

B.2.2 How To Evaluate the Exactly Graph Matching? We utilize SMILES strings of molecule graphs for graph matching. SMILES (Simplified Molecular Input Line Entry System) [64] is a widely used notation in chem-informatics for encoding the structure of chemical molecules using short ASCII strings, thereby facilitating efficient representation and manipulation of chemical structures. The reconstructed graph is matched with the target graph if their SMILES strings are identical [36, 54]. Specifically, let $\mathcal{T}_r = \{\tau_1, \ldots, \tau_m\}$ and $\mathcal{T}_t = \{\tau_1, \ldots, \tau_m\}$ denote the SMILES strings sets of reconstructed graphs $\mathcal{D}_r$ and training graphs $\mathcal{D}_t$, respectively. Then, Reconstruction rate $= |\mathcal{T}_r \cap \mathcal{T}_t|/|\mathcal{T}_r|$, where $|\mathcal{T}_r \cap \mathcal{T}_t|$ denotes the number of the SMILES strings of reconstructed graphs exactly matched with those of graphs in $\mathcal{D}_t$. This method is efficient and reasonable for matching molecule graphs as SMILES strings cover various aspects of molecular structure, encoding information about atoms, bonds, and connectivity within a molecule.

C ADDITIONAL RESULTS OF THE IMPACT OF SELECTED GRAPHS

We report the validity and reconstruction rate on FreeSolv in Fig. 7. Moreover, we also report the absolute number of the reconstructed graphs of GraphSteal on FreeSolv and QM9 in Tab. 3. The observations are similar to those of Fig. 4 in Sec. 5.3.

D IMPACT OF THE TRAINING/AUXILIARY SET

D.1 Training/Auxiliary Set Distribution Shift

To investigate performance where there is a distribution shift between the auxiliary and target training datasets, we propose to divide QM9 in the following way to make the auxiliary and target training datasets have a distribution shift. Specifically, we first train a GCN with graph labels and use the trained GCN to obtain graph representations. We then apply K-Means to cluster these graphs into 8 groups, setting 1 group as the training set and the remaining groups as the auxiliary set. Hence, there is a significant distribution shift between the two sets. The results on the QM9 dataset are reported in Tab. 5. From the table, we can observe that GraphSteal still outperforms the baselines in this setting, demonstrating its effectiveness in the case of there is a significant distribution shift between the training and auxiliary datasets.

D.2 Training/Auxiliary Set Split Ratio

In this section, we investigate the impact of the training/auxiliary dataset split (T/A split). We set the T/A split at {10%/70%, 50%/30%, 60%/20%}. Specifically, 50%/30% means that 50% and 30% of the original dataset serve as the training set for the GNN classifier and the auxiliary set, respectively. The target GNN model is set as GCN. All other settings are the same as that in Sec. 5.3. The results on QM9 are reported in Tab. 6. From the table, we observe that:

- GraphSteal achieves good performance across all T/A split settings. This demonstrates the effectiveness of GraphSteal in various T/A split ratios, especially when the size of the auxiliary set is small.
- As the T/A split ratio increases, the reconstruction rate rises. We analyze that the reason is that although the size of the auxiliary dataset decreases as the T/A split ratio increases, it remains sufficient for the graph diffusion model to acquire enough knowledge to generate novel and valid graphs for the reconstruction selection to identify the generated graphs that belong to the target training graphs. Moreover, as the size of the training dataset also increases, our framework can better match the reconstructed graphs with more training graphs, leading to an increase in the reconstruction rate.

We further fix the training dataset at 10% of the original dataset and vary the size of the auxiliary dataset to {20%, 30%, 70%} of the original dataset. The results on QM9 are reported in Tab. 7. From the table, we observe that GraphSteal consistently achieves good performance across all auxiliary dataset sizes, further validating the effectiveness of GraphSteal in various auxiliary dataset sizes.

E POTENTIAL COUNTERMEASURES

In this section, we explore the potential countermeasures of graph stealing attacks. Since graph stealing attacks represent a new type of privacy attack that steals private training graphs from trained GNN classifiers, there is no existing work studying the defense against this attack. One potential defense could be differential privacy (DP). Here, we investigate the effectiveness of DP against graph stealing attacks. Following [1], we add Gaussian noise to the gradients in each training iteration of target GNN classifier training to ensure $(\epsilon, \delta)-$ DP. We fix $\delta = 10^{-5}$ and vary the noise scale to $\{1.0, 5.0, 10.0\}$. The reconstruction results on QM9 are reported in Tab. 4. From the table, we can observe that as the privacy budget ϵ decreases, GraphSteal consistently shows good validity, uniqueness, and FCD but only a slight drop in the reconstruction rate. This indicates that applying differential privacy to GNNs cannot prevent graph stealing attacks effectively. Therefore, there is an emerging need to design more effective countermeasures against GraphSteal.

F ETHICAL IMPLICATIONS

In this paper, we study a novel privacy attack problem of extracting private training graphs from the trained GNN. Our work uncovers the vulnerability of GNNs to the graph stealing attack and discusses potential countermeasures against this attack. Our work aims to raise awareness about the privacy issues inherent in GNNs and inspire the following works to develop more advanced privacy-preserving methods to protect against graph stealing attacks. All datasets we used in this paper are publicly available, no sensitive

Table 4: Reconstruction results of GraphSteal against GCN trained with differential privacy on QM9 dataset.

Metrics	$\epsilon = 1.0$	$\epsilon = 5.0$	$\epsilon = 10.0$	no DP
Validity (%) ↑	98.6±1.1	99.2±1.1	98.8±1.3	98.8±0.8
Uniqueness (%) ↑	100±0	100±0	100±0	100±0
Recon. Rate (%) ↑	22.4±2.4	24.0±3.6	26.0±0.7	29.2±3.1
FCD (%) ↓	2.3±0.1	2.3±0.2	2.1±0.2	2.0±0.2

Table 5: Reconstruction results on QM9 for GCN when training and auxiliary sets have a significant distribution shift.

Metrics	Valid. (%)	Unique. (%)	Recon. (%)	FCD (%)
BL-Rand	3.2±1.1	**100±0**	0±0	18.8±0.8
BL-Diff	**100±0**	99.6±0.5	1.8±0.8	9.0±2.4
GraphMI-G	10.0±9	**100±0**	0±0	11.5±0.5
Ours	**100±0**	**100±0**	**7.0±1.8**	**6.2±0.3**

Table 6: Impact of the T/A split on QM9 dataset.

T/A split	10%/70%	50%/30%	60%/20%
Validity (%) ↑	98.8±0.8	99.0±0.7	99.4±0.5
Unique. (%) ↑	100±0	98.8±0.4	100±0
Recon. (%) ↑	29.2±3.1	45.0±1.8	60.6±4.0
FCD (%) ↓	2.0±0.2	1.8±0.3	1.9±0.3

Table 7: Impact of the size of the auxiliary dataset on QM9 dataset.

Auxiliary size	20%	30%	70%
Validity (%) ↑	99.2±0.7	99.2±0.8	98.8±0.8
Unique. (%) ↑	100±0	100±0	100±0
Recon. (%) ↑	23.2±2.0	25.1±2.3	29.2±3.1
FCD (%) ↓	1.8±0.2	1.7±0.3	2.0±0.2

or private dataset from individuals or organizations was used. Our work is mainly for research purposes and complies with ethical standards. Therefore, it does not have any negative ethical impact on society.