Unveiling the Secrets without Data: Can Graph Neural Networks Be Exploited through Data-Free Model Extraction Attacks?

Yuanxin Zhuang, Chuan Shi, and Mengmei Zhang, *Beijing University of Posts and Telecommunications;* Jinghui Chen, *The Pennsylvania State University;* Lingjuan Lyu, *SONY AI;* Pan Zhou, *Huazhong University of Science and Technology;* Lichao Sun, *Lehigh University*

https://www.usenix.org/conference/usenixsecurity24/presentation/zhuang

This paper is included in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Open access to the Proceedings of the 33rd USENIX Security Symposium is sponsored by USENIX.

Unveiling the Secrets without Data: Can Graph Neural Networks Be Exploited through Data-Free Model Extraction Attacks?

Yuanxin Zhuang Chuan Shi * Mengmei Zhang
Beijing University of Posts and Telecommunications
Key Laboratory of Trustworthy Distributed Computing and Service (BUPT), Ministry of Education
Jinghui Chen
The Pennsylvania State University
Lingjuan Lyu
SONY AI
Pan Zhou
Huazhong University of Science and Technology
Lichao Sun
Lehigh University

Abstract

Graph neural networks (GNNs) play a crucial role in various graph applications, such as social science, biology, and molecular chemistry. Despite their popularity, GNNs are still vulnerable to intellectual property threats. Previous studies have demonstrated the susceptibility of GNN models to model extraction attacks, where attackers steal the functionality of GNNs by sending queries and obtaining model responses. However, existing model extraction attacks often assume that the attacker has access to specific information about the victim model's training data, including node attributes, connections, and the shadow dataset. This assumption is impractical in real-world scenarios. To address this issue, we propose STEALGNN, the first data-free model extraction attack framework against GNNs. STEALGNN advances prior GNN extraction attacks in three key aspects: 1) It is completely *data-free*, as it does not require actual node features or graph structures to extract GNN models. 2) It constitutes a *full-rank attack* that can be applied to node classification and link prediction tasks, posing significant intellectual property threats across a wide range of graph applications. 3) It can handle the most challenging *hard-label* attack setting, where the attacker possesses no knowledge about the target GNN model and can only obtain predicted labels through querying the victim model. Our experimental results on four benchmark graph datasets demonstrate the effectiveness of STEALGNN in attacking representative GNN models.

1 Introduction

The success of GNNs heavily relies on large-scale datasets, similar to deep learning algorithms for images and texts. These datasets often contain sensitive information collected from users [38, 41], enabling the development of powerful GNN models in critical domains such as healthcare [18], banking systems [37], and bioinformatics [17]. For example, GNNs have been utilized to analyze brain networks using fMRI data [17]. Furthermore, GNN model owners may offer query API services, allowing others to access and benefit from the knowledge acquired by their GNN models. Pre-trained GNN models are also commonly shared with third parties for various downstream tasks [20, 22, 43, 44]. However, the availability of these pretrained models, combined with the sensitive data they were trained on, raises privacy concerns.

Table 1: Comparison with Existing Model Extraction Methods for GNNs.

Method	Data-Free	Node Classification		Link Prediction
		Transductive	Inductive	
DeFazio et al.'s work [4]		✓		
Wu et al.'s work [39]		✓		
Shen et al.'s work [33]			✓	
STEALGNN	✓	✓	✓	✓

General model extraction attacks. Model extraction attacks are designed to steal information about a model's architecture and parameters. These attacks can either reconstruct the original model or create a substitute model that performs similarly [13, 33]. In these attacks, the attacker crafts multiple queries to be input into the victim model's API and captures the corresponding output. Subsequently, a local model is trained using this paired data (input, output). As real data for queries is often limited, recent studies [14, 35] have explored data-free model extraction attacks. These methods employ a generator to synthesize query examples as input to the victim model. The generator and surrogate model are trained in a competitive manner, similar to the principles of Generative Adversarial Networks (GANs) [1, 10]. The surrogate model aims to mimic the predictions of the victim model, while the generator maximizes the disagreement between the victim model's predictions and the surrogate model's predictions. It is important to note that most existing efforts on data-free model extraction attacks primarily focus on the Computer Vision (CV) domain [21]. However, developing efficient and effective data-free model extraction attacks for GNN models remains an open challenge, primarily due to the involvement of both graph structure and node attributes in GNN modeling.

*corresponding author

Limitations of current model extraction attacks against GNNs. Current model extraction attacks on GNNs are designed based on different levels of background knowledge available to the attackers [4,33,34,39]. For instance, attackers who have access to the node attributes of a set of adversarial nodes can generate queries from these adversarial nodes and utilize the responses to train a substitute GNN. One example of such an attack is presented in [39], where discrete graph structure learning methods (e.g., LDS [8]) are employed to construct a connected substitute graph using these node attributes. Another approach, as described in [33], involves initializing the graph structure using K-nearest neighbors (KNN) based on the node attributes and then updating it using a graph structure learning framework [2].

The availability of information about the real training data is crucial for launching a successful model extraction attack in all these scenarios. However, it is important to note that most real-world graphs, such as social and economic networks, involve human-related activities. Due to privacy concerns, these graphs are typically highly protected and not accessible to potential attackers. For example, in a social network, user-related information like friend lists, profiles, likes, comments, and other details are considered highly sensitive and private. As a result, existing attacks become ineffective without access to the actual training dataset.

Moreover, prior research has predominantly concentrated on validating the effectiveness of these attacks within the context of node classification tasks, largely overlooking the critical evaluation of their applicability to another crucial real-world scenario: link prediction tasks. Node classification tasks involve predicting labels or attributes for individual nodes within a graph, whereas link prediction tasks revolve around forecasting the presence or absence of edges between pairs of nodes. Both of these tasks play pivotal roles in various domains such as social sciences, biology, and molecular chemistry. Hence, it becomes imperative to ascertain whether model extraction attacks can effectively compromise GNNs when deployed for link prediction tasks. Tab. 1 provides a comparative overview of StealGNN in comparison to existing methods, shedding light on the specific scenarios and experimental tasks for which they were designed. StealGNN conducted experiments to assess the performance of both node classification and link prediction tasks. This comprehensive evaluation enables us to offer a more holistic understanding of the attack's potential impact across a broad spectrum of graph applications. Notably, the studies by [4] and [39] primarily focused on experimentation with transductive GNNs, which are designed to make predictions exclusively for nodes present during training, while [33] concentrated on inductive GNNs, which possess the capability to predict previously unseen nodes during inference.

Challenges of conducting data-free model extraction attacks on GNNs. Data-free model extraction attacks against GNNs present unique challenges when compared to data-free attacks on other neural networks: (1) Graphs exhibit diverse levels of connectivity and hierarchical organization, making it difficult to generate graphs with specific structural properties such as connectivity patterns, node degrees, or community structures. (2) Graphs often include attributes associated with nodes and edges in addition to structural information. Generating meaningful attributes while ensuring their consistency with the graph structure adds an additional layer of complexity. This complexity arises from the need to create attributes that not only carry relevant information but also align seamlessly with the underlying graph topology and relationships, maintaining coherence and relevance throughout the entire graph analysis process. (3) Unlike pixel-level image data that allows for smooth spatial transformations, graph structures are discrete. This discrete nature poses challenges when applying traditional gradient-based optimization methods like backpropagation.

Our contribution. This paper focuses on data-free model extraction attacks, which is considered the most challenging and widely deployed setting for such attacks on GNNs. Importantly, StealGNN does not require access to real-world data. Instead, it utilizes a generative model to generate synthetic graphs for launching the attack. A key innovation of StealGNN lies in its ability to conduct model extraction without the need for access to real-world graph data. It leverages a powerful graph generator during the training process to create synthetic graphs for launching the attack. As illustrated in Fig. 1, the attacker employs this innovative generator to produce synthetic graphs specifically crafted for querying the victim model. The surrogate model is trained to imitate the victim model based on the input-output pair.

To address the challenge of lacking access to actual gradients of the victim model for updating the generator's parameters, we categorize the attacks based on the training scheme of the generator into three types: **(1) Type I Attack:** In this type of attack, gradients are propagated back through a surrogate model and the gradient approximation of the victim model. By leveraging the estimated gradients of the victim model and the surrogate model's gradients, the generator optimizes its parameters to capture knowledge from both models. **(2) Type II Attack:** This type of attack involves propagating gradients back through only one surrogate model. While this strategy simplifies the gradient flow, it still allows the generator to learn from the surrogate model. **(3) Type III Attack:** In this type of attack, gradients are passed back through two surrogate models. By utilizing the gradients derived from both surrogate models, the generator benefits from diverse perspectives and insights, enhancing the learning and knowledge extraction process. Extensive experimentation has consistently demonstrated the exceptional performance of StealGNN across all three attack scenarios. Our contributions can be summarized as follows:

(1) StealGNN: We introduce StealGNN, a pioneering research endeavor that addresses data-free model extraction

attacks on Graph Neural Networks (GNNs). By developing a trainable graph generator, STEALGNN demonstrates its ability to train a high-quality surrogate model from black-box GNN models, even when only hard labels are available.

(2) Systematic Study: We conduct a comprehensive investigation and propose three extraction attack scenarios based on the updating scheme of the graph generator parameters. These scenarios provide a systematic understanding of different approaches within the STEALGNN framework.

(3) Task Flexibility: STEALGNN is highly adaptable and robust, capable of effectively handling various tasks. It can be applied to both node classification and link prediction tasks, showcasing its versatility. The results obtained from our experiments provide compelling evidence for the superior performance of STEALGNN in these tasks.

2 Background

2.1 Notations

We define an undirected, unweighted, and attributed graph as $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, where $\mathbf{V}$ represents the set of nodes, and $\mathbf{E}$ represents the set of edges. The adjacency matrix $\mathbf{A}$ is defined as follows: $\mathbf{A}_{u,v} = 1$ if the edge $(u,v) \in \mathbf{E}$, and 0 otherwise. In this notation, n denotes the total number of nodes, d represents the dimension of the node features, and c indicates the number of node classes. We use $\mathbf{F} \in \mathbb{R}^{n \times d}$ to denote the node feature matrix, and $\mathbf{Y} \in \mathbb{R}^{n \times c}$ represents the node label matrix. Furthermore, we refer to the graph generator as $\mathcal{M}_G$, the feature generator as $\mathcal{M}_{G_F}$, the structure generator as $\mathcal{M}_{G_A}$, the victim GNN model as $\mathcal{M}_V$, and the surrogate GNN model as $\mathcal{M}_S$.

2.2 Graph Neural Networks

Graph Neural Networks (GNNs) have garnered significant attention in the field of graph-structured data learning for a variety of tasks [40]. In general, most GNN models follow a paradigm that includes neighborhood aggregation and message passing [9]. At the k-th layer, the representation $\mathbf{h}_v^k$ of each node v is learned by iteratively aggregating the embeddings $\mathbf{h}_u^{k-1}$ of its neighboring nodes $u \in \mathcal{N}(v)$:

$$\mathbf{h}_v^k = \text{AGG}\left(\mathbf{h}_v^{k-1}, \text{MSG}\left(\mathbf{h}_v^{k-1}, \mathbf{h}_u^{k-1}\right)\right), u \in \mathcal{N}(v), \quad (1)$$

where $\mathbf{h}_v^0 = \mathbf{X}_v$ represents the node feature of node v. MSG stands for message aggregation. It calculates the messages exchanged between a node v and its neighbors u at layer $k-1$. These messages typically encapsulate information about local structural patterns and features. AGG is referred to as the aggregation function. It combines information from the previous layer's representation $\mathbf{h}_v^{k-1}$ and the messages generated by the MSG function, which involves interactions between the node of interest v and its neighboring nodes $u \in \mathcal{N}(v)$. After K iterations of aggregation, a node representation, denoted as $\mathbf{h}_v^K$, captures the structural information within its K-hop network neighborhood.

3 Threat Model

3.1 Attacker's Setting

Generally, the knowledge of an attacker regarding the threat model in GNN-based attacks can be classified into two settings: black-box attack and gray-box attack. (1) Black-Box Attack: The attacker lacks access to the target GNN's parameters, architecture details, and hyperparameters. They can query the model for predictions. (2) Gray-Box Attack: In this scenario, the attacker has partial knowledge about the target model, like its structure, but no access to parameters.

In this paper, we address the most challenging adversarial setting: black-box attacks. Our attack methodology fundamentally differs from previous black-box attacks on GNNs [33, 39]. These attacks assume that the adversary has access to the victim model's original training data, such as node features, structure, or shadow datasets. However, this assumption often misaligns with real-world scenarios, where training data is frequently treated as private and diligently protected from potential attackers. For instance, a real-world deployment of a GNN in a financial institution for fraud detection. The GNN is trained on a vast dataset containing transaction histories, user profiles, and behavioral patterns. In this context, the training data is highly confidential and sensitive, as it may contain personal information and proprietary insights. Allowing external access to this training data would not only violate privacy regulations but also pose substantial security risks. Given such real-world example and the broader context of privacy concerns, the assumption that adversaries have access to the victim model's training data becomes increasingly unrealistic. Therefore, our work addresses a more realistic scenario wherein attackers are constrained to rely on synthetic datasets to query the victim model and extract valuable information.

3.2 Attacker's Goal

The objective of this attack is to extract information from the target model by training a model that exhibits similar behavior. The attack may focus on different aspects of the target model's information, leading to two primary goals in model extraction: (1) The attacker aims to obtain a model that achieves comparable accuracy to the target model. (2) The attacker attempts to replicate the decision boundary of the target model. Model extraction attacks pose a threat to the security of models used in API services [26] and can serve as a foundation for various privacy attacks and adversarial attacks.

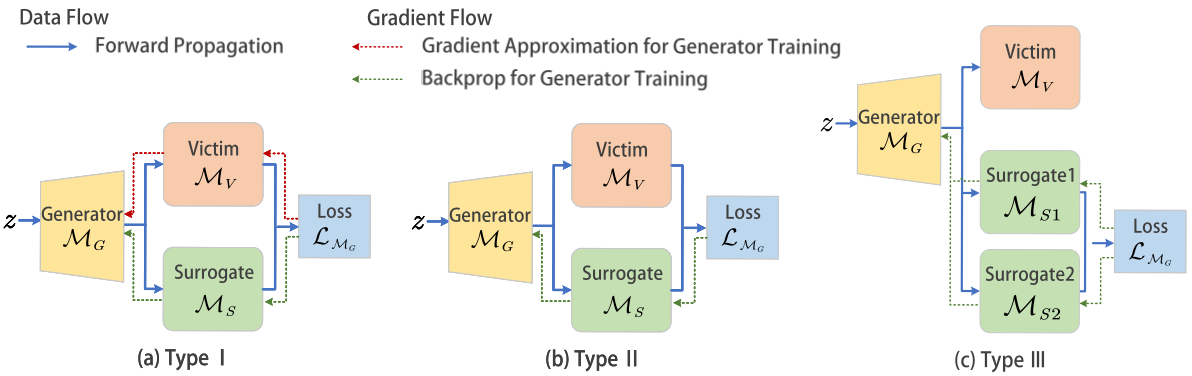


Figure 1: Generator update strategies: (a) Gradients are propagated through both a surrogate model and an approximation of the victim model. This approach aims to capture a broader understanding of the victim model's decision-making process. (b) Gradients are passed back through a single surrogate model. This simplifies the update process by relying on the knowledge learned by a single surrogate model, potentially sacrificing some diversity in the learned knowledge. (c) Gradients are passed back through two surrogate models. This allows the generator to benefit from the diverse perspectives captured by both surrogate models, enhancing its ability to emulate the victim model more effectively.

3.3 Attacker's Taxonomy

The computation of the model extraction loss is essential for updating the parameters of the local graph generator through backpropagation. In typical scenarios, gradients can be backpropagated to the generator by utilizing both a victim model and a surrogate model. However, in black-box scenarios, the parameters of the victim model are inaccessible, rendering it impossible to backpropagate gradients through the victim model. Due to this constraint, we have developed three types of attacks based on different sources of gradients for updating the generator's parameters.

Type I Attack: As illustrated in Fig. 1(a), in this type of attack, the gradient is propagated back through the surrogate model and estimated using a gradient approximation method for the victim model. The zeroth-order gradient estimate, introduced by Polyak and Juditsky [28], is a technique used to approximate gradients when the analytical form of the model is unavailable. Instead, it relies on querying the model at various input points and observing the corresponding output values to estimate the gradient. This approach is particularly valuable in the context of black-box optimization problems. By iteratively sampling and comparing model predictions, the zeroth-order gradient estimate enables derivative-free optimization and exploration of the model's behavior.

By leveraging the estimated gradient of the knowledge-rich victim model, the generator can enhance its graph generation process to capture and incorporate the valuable knowledge from the victim model into the agent model. This optimization allows the generator to produce graphs that effectively encapsulate the desired knowledge and transfer it to the agent model, thereby improving the performance and capabilities of the agent model.

Type II Attack: As illustrated in Fig. 1(b), in this type of attack, the generator is trained exclusively using gradients propagated back through the surrogate model. As the model extraction process progresses, the surrogate model refines its behavior, gradually aligning with that of the victim model. This alignment enables the gradients derived from the surrogate model to serve as a reasonable approximation of the gradients of the victim model. Leveraging this approximation, the generator can effectively train and optimize itself, facilitating the extraction of knowledge from the victim model and its subsequent incorporation into the agent model.

Type III Attack: When the graph generated by the generator passes through both the victim model and the surrogate model, two possible outcomes can occur: consistent predicted categories between the victim and surrogate models or inconsistent predicted categories. If the generated graph yields consistent predicted categories, it does not contribute to the imitation of the surrogate model from the victim model. As a result, the surrogate model fails to gain additional information that can improve its emulation of the victim model. However, if the generated graph results in inconsistent predictions between the two models, the surrogate model can acquire deeper and more complex knowledge from the victim model, gradually approaching the decision-making process of the victim model.

As illustrated in Fig. 1(c), in this attack, we simultaneously train two surrogate models and introduce a loss function that enforces inconsistency in their predictions. When the same graph is inputted to both surrogate models and the victim model, the inconsistent predictions of the surrogate models indicate that at least one of them has learned valuable information inconsistent with the victim model. This approach prevents the generator from solely generating simple graphs to minimize the model extraction loss, as the surrogate model would fail to learn any meaningful information.

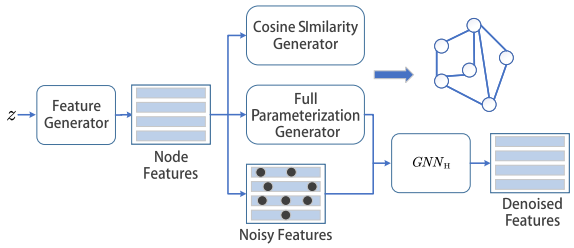


Figure 2: The framework for graph generators. It consists of two types of graph structure generators: the cosine similarity generator and the full parameterization generator.

Advantages, Disadvantages, and Real-World Applicability of Attack Types: In practical scenarios, the choice of attack type hinges on various factors, including the attacker's constraints on querying the victim model, available computational resources, and the depth and complexity of knowledge extraction objectives. For a more detailed connection with real-world scenarios, please refer to the appendix A.4.

(1) Type I Attack: This method effectively harnesses gradient estimation techniques, making it well-suited for intricate proprietary models. However, it may introduce noise and consume substantial computational resources, especially when numerous queries are required to obtain accurate gradient estimates. Type I attacks excel when multiple queries to the victim model are feasible. (2) Type II Attack: These attacks solely rely on surrogate model gradients, simplifying their implementation. However, compared to Type I attacks, Type II knowledge transfer may be slower, given the initial disparity between surrogate and victim model gradients. This approach proves advantageous when query access to the victim model is limited (e.g., due to API rate restrictions) or when the victim model's behavior evolves gradually. Type II attacks are computationally less intensive than Type I but remain effective in accumulating knowledge over time. (3) Type III Attack: Type III attacks aim to capture intricate and profound knowledge from victim models. However, their implementation involves the complexity of deploying and training two surrogate models, along with a loss function to enforce inconsistency. Simultaneously running and monitoring these two surrogate models can be computationally demanding. This approach is suitable when attackers seek to ensure that surrogate models acquire in-depth knowledge from the victim model, particularly in scenarios where simplistic attacks may prove insufficient or when the victim model resists direct extraction attempts.

4 StealGNN

4.1 Attack Framework

Given a victim model $\mathcal{M}_V(\mathbf{G};\theta_V)$, the objective of StealGNN is to learn a surrogate model $\mathcal{M}_S(\mathbf{G};\theta_S)$ that closely resembles the victim model in terms of its behavior and predictions. To achieve this, a graph generator $\mathcal{M}_G(z;\theta_G)$ is employed to generate synthetic graphs $\mathbf{G}$. The generator is updated based on the loss function $\mathcal{L}_{\mathcal{M}_G}$. The surrogate model $\mathcal{M}_S(\mathbf{G};\theta_S)$ is trained using input-output pairs $(\mathbf{G}, \mathcal{M}_V(\mathbf{G}))$, where $\mathcal{M}_V(\mathbf{G})$ represents the output of the victim model for a given input graph $\mathbf{G}$. By training the surrogate model based on these pairs, the surrogate model learns to approximate the behavior and predictions of the victim model.

The proposed StealGNN attack follows a specific data flow, as shown in Fig. 1(a) with a blue arrow. It begins by sampling a random noise vector z from a standard normal distribution. The graph generator $\mathcal{M}_G$ takes this noise vector as input and generates a graph $\mathbf{G}$. This generated graph is then used as input for both the victim model $\mathcal{M}_V$ and the surrogate model $\mathcal{M}_S$, which perform their respective inferences and produce output predictions. The loss function is computed based on the outputs of the victim model $\mathcal{M}_V$ and the surrogate model $\mathcal{M}_S$, measuring the discrepancy between their predictions. This loss function serves as a measure of how well the surrogate model can emulate the victim model.

We will now provide a detailed description of the graph generator's architecture and explain the training process for both the generator and surrogate model.

4.2 Graph Generator

4.2.1 Feature Generator

The feature generator $\mathcal{M}_{G_F}(z)$ is a function defined as $\mathcal{M}_{G_F}: \mathbb{R}^{1\times z} \to \mathbb{R}^{n\times d}$. It transforms a one-dimensional noise vector z into a two-dimensional node feature matrix $\mathbf{F}$ with n rows (representing the number of nodes in the graph) and d columns (representing the dimension of the node features). The purpose of this transformation is to ensure that the generated graph's node features align with the expected input feature dimension of the victim model. It is worth noting that since the victim model is fixed, the required input feature dimension can be determined from the specifications of the victim model. Additionally, the number of nodes is a hyperparameter of the generator, allowing the attacker to choose the desired size for the generated number of nodes. In Fig. 6, we provide experimental results illustrating the outcomes of different choices for this hyperparameter.

4.2.2 Structure Generator

The structure generator $\mathcal{M}_{G_A}(\mathbf{F})$ is a function $\mathcal{M}_{G_A}: \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times n}$ with parameters θ_A which takes the node features $\mathbf{F} \in \mathbb{R}^{n\times d}$ as input and produces a matrix $\mathbf{A} \in [0,1]^{n\times n}$ as output. Specifically, $\mathbf{A}_{ij} \in [0,1]$ indicates the presence of an edge between nodes $\mathbf{x}_i$ and $\mathbf{x}_j$. In order to conduct a comprehensive analysis and promote flexibility and diversity during the graph structure generation process, we have designed two distinct graph structure generators. These generators are as follows:

Cosine similarity graph structure generator: One approach to infer a graph structure is to utilize a similarity metric and assign the edge weight between two nodes based on their similarity [30]. In order to achieve sparsity in the graph structure, we construct a similarity graph using the cosine similarity of the node features. This approach only connects pairs of nodes whose similarity exceeds a predefined threshold. To be more specific, given the node features $\mathbf{F}$, the element of the adjacency matrix $\mathbf{A}_{cos}$ can be computed as follows:

$$\mathbf{A}_{cos_{ij}} = \text{Cosine}(\mathbf{F}_i, \mathbf{F}_j) = \frac{\mathbf{F}_i \cdot \mathbf{F}_j}{\max\left(\|\mathbf{F}_i\|_2 \cdot \|\mathbf{F}_j\|_2\right)}. \quad (2)$$

To promote sparsity in the learned adjacency matrix $\mathbf{A}_{cos}$ and reduce the impact of small values on the aggregation process in GNNs, we adopt a thresholding mechanism. Entries in $\mathbf{A}_{cos}$ with values smaller than a given threshold τ are removed. Specifically, we set a threshold value τ, and if $\mathbf{A}_{cos_{ij}}$ is larger than τ, it is set to 1, indicating the presence of an edge between nodes $\mathbf{x}_i$ and $\mathbf{x}_j$. Otherwise, it is set to 0, indicating the absence of an edge between the corresponding nodes.

Full parameterization graph structure generator: In this generator, we employ the Full parameterization method [8], which is also utilized in the SLAPS framework [7]. Given the parameter matrix $\theta_A \in \mathbb{R}^{n\times n}$, the generator function is defined as $\tilde{\mathbf{A}}_{fp} = \mathcal{M}_{G_A}(\mathbf{F};\theta_A) = \theta_A$. In other words, the generator disregards the input node features and directly optimizes the adjacency matrix. The full parameterization in this generator is simple and flexible, enabling the learning of any adjacency matrix. Once the parameterized adjacency matrix is generated, it undergoes additional processing to obtain a well-defined symmetric adjacency matrix. Subsequently, self-supervised learning is employed to train the learnable parameters within the structure generator.

(1) Adjacency processor: To address the issues of non-symmetry, non-normalization, and potential negative values in the generated adjacency matrix $\tilde{\mathbf{A}}_{fp}$, we apply a series of transformations to obtain the final adjacency matrix $\mathbf{A}_{fp}$. Specifically, we define $\mathbf{A}_{fp}$ as follows: $\mathbf{A}_{fp} = \frac{1}{2}\boldsymbol{D}^{-\frac{1}{2}}\left(\text{P}(\tilde{\mathbf{A}}_{fp}) + \text{P}(\tilde{\mathbf{A}}_{fp})^T\right)\boldsymbol{D}^{-\frac{1}{2}}$. In this formulation, P represents an activation function that is applied element-wise to its input, ensuring non-negativity. The sub-expression $\frac{1}{2}\left(\text{P}(\tilde{\mathbf{A}}_{fp}) + \text{P}(\tilde{\mathbf{A}}_{fp})^T\right)$ guarantees that the resulting matrix $\text{P}(\tilde{\mathbf{A}}_{fp})$ is symmetric. Finally, to normalize the resulting symmetric adjacency matrix, we compute its degree matrix $\boldsymbol{D}$ and multiply it from the left and right with $\boldsymbol{D}^{-\frac{1}{2}}$. By applying these transformations, we obtain a symmetric, non-negative, and normalized adjacency matrix $\mathbf{A}_{fp}$. This ensures the compatibility and suitability of the generated adjacency matrix for further processing and utilization in the model extraction process.

(2) Self-supervision: This self-supervised task is inspired by denoising autoencoders [36]. We define $\text{H} : \mathbb{R}^{n\times f} \times \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times f}$ as a GNN with parameters θ_H. It takes the node features and the generated adjacency matrix as input and produces updated node features of the same dimension as output. We train H to denoise the input features by feeding it a noisy version $\tilde{\mathbf{F}}$ of the original features $\mathbf{F}$ and expecting it to generate the denoised features $\mathbf{F}$. During training, we minimize the following loss function:

$$\mathcal{L}_\text{denoise} = \frac{1}{|\mathbf{F}_{idx}|}\sum_{i\in\mathbf{F}_{idx}} \mathcal{L}_\text{reconstruction}\left(\mathbf{F}_i, \text{H}\left(\tilde{\mathbf{F}}, \mathbf{A}_{fp}; \boldsymbol{\theta}_\text{H}\right)_i\right), \quad (3)$$

where idx represents the indices corresponding to the elements of $\mathbf{F}$ to which we introduce noise. Similarly, $\mathbf{F}_{idx}$ refers to the values at these selected indices. In each epoch, idx consists of a random uniform selection of r percent of the indices from $\mathbf{F}$. The noise is introduced by replacing the values at the indices specified by idx with 0. $\mathcal{L}_\text{reconstruction}$ denotes the reconstruction loss between the denoised features $\mathbf{F}_i$ and the corresponding predicted features $\text{H}(\tilde{\mathbf{F}}, \mathbf{A}_{fp})_i$. We use the mean-squared error loss. In this attack, the generator initially optimizes the feature generator and structure generator parameters using $\mathcal{L}_\text{denoise}$. This optimization aims to minimize the denoising loss, which encourages the GNN H to learn meaningful representations capable of reconstructing the original features accurately, even in the presence of noise. This self-supervised training process plays a crucial role in capturing informative and meaningful graph structures. Subsequently, the resulting graph is fed into both the victim and agent models, facilitating the extraction of knowledge from the victim model.

4.3 Training Surrogate Model

The surrogate model $\mathcal{M}_S$ is trained using the responses obtained from the victim model $\mathcal{M}_V$ when provided with a query graph $\mathbf{G}$. Due to the unavailability of the victim model, the attacker encounters different types of outputs when querying the victim model, which can range from probability outputs (referred to as soft labels) to one-hot label predictions (hard labels). This paper specifically focuses on studying the most challenging scenario, where the victim model exclusively provides hard labels as output.

The generator $\mathcal{M}_G$ takes a noise vector z as input and generates a graph $\mathbf{G}$ (as described in Equation 4). This generated graph $\mathbf{G}$ is then used to query both $\mathcal{M}_V$ and $\mathcal{M}_S$, resulting in the outputs $\overrightarrow{\mathbf{Y}_V}$ and $\overrightarrow{\mathbf{P}_S}$ respectively (as shown in Equation 5). It is important to note that $\overrightarrow{\mathbf{Y}_V}$ represents hard labels obtained from the victim model, while $\overrightarrow{\mathbf{P}_S}$ represents soft labels derived from the surrogate model.

$$\mathbf{G} = \mathcal{M}_G(z;\theta_G); \quad z \sim \mathcal{N}(0, I), \quad (4)$$

$$\overrightarrow{\mathbf{Y}_V} = \mathcal{M}_V(\mathbf{G},\theta_V); \quad \overrightarrow{\mathbf{P}_S} = \mathcal{M}_S(\mathbf{G};\theta_S). \quad (5)$$

The surrogate model parameters θ_S are updated using the cross-entropy loss function [3] to minimize the discrepancy between $\overrightarrow{\mathbf{P}_S}$ and $\overrightarrow{\mathbf{Y}_V}$:

$$\mathcal{L}_{\mathcal{M}_S} = CE(\mathcal{M}_S(\mathbf{G};\theta_S), \mathcal{M}_V(\mathbf{G};\theta_V)) = -\sum_{i=1}^{n}\overrightarrow{\mathbf{Y}_{V_i}}\log\overrightarrow{\mathbf{P}_{S_i}}. \quad (6)$$

By training the surrogate model $\mathcal{M}_S$ using the graph $\mathbf{G}$ and the ground truth labels $\overrightarrow{\mathbf{Y}_V}$ of the victim model $\mathcal{M}_V$, we can extract the knowledge embedded in $\mathcal{M}_V$. This is achieved by guiding $\mathcal{M}_S$ to learn the exact mapping of $\mathcal{M}_V$, thereby capturing its knowledge and replicating its behavior.

4.4 Training Graph Generator

To mitigate the issue of simplistic or duplicate graphs generated by the generator, we introduce a mechanism to encourage the production of more complex and diverse graphs. This mechanism aims to increase the discrepancy between the surrogate and victim models. Specifically, the generator model $\mathcal{M}_G$ is trained to maximize the disagreement between the predictions of the victim model $\mathcal{M}_V$ and the surrogate model $\mathcal{M}_S$. In other words, the loss function for the generator in the Type I Attack and Type II Attack is defined as $\mathcal{L}_{\mathcal{M}_G} = -\mathcal{L}_{\mathcal{M}_S}$. This formulation incentivizes the generator to produce diverse and complex graphs, thereby amplifying the discrepancy between the two models.

4.4.1 Type I Attack

Since the output of the victim model consists of hard labels, which are discrete and non-probabilistic predictions, it cannot be directly utilized to compute approximate gradients. To address this, we employ a label-smoothing regularization technique [23], which aims to soften the hard-label outputs of the victim model into soft labels. This soft label output is then utilized to approximate the gradient of the victim model, enabling us to perform gradient-based optimization. As shown in Fig. 1(a), our objective is to update the parameters θ_G of the generator model using gradient descent in order to minimize the loss function $\mathcal{L}_{\mathcal{M}_G}$ as depicted in Equation 7:

$$\theta_G^{t+1} = \theta_G^t - \eta \nabla_{\theta_G} \mathcal{L}_{\mathcal{M}_G}. \tag{7}$$

To update θ_G, we need to compute the derivative of the loss function $\nabla_{\theta_G} \mathcal{L}_{\mathcal{M}_G}$. This derivative can be decomposed into two components, as shown in Equation 8.

$$\nabla_{\theta_G} \mathcal{L}_{\mathcal{M}_G} = \frac{\partial \mathcal{L}_{\mathcal{M}_G}}{\partial \theta_G} = \frac{\partial \mathcal{L}_{\mathcal{M}_G}}{\partial \mathbf{G}} \times \frac{\partial \mathbf{G}}{\partial \theta_G}. \tag{8}$$

The second component corresponds to the derivative of the generator's output with respect to its parameters, which can be easily computed by performing backpropagation through $\mathcal{M}_G$. However, the first component, which represents the derivative of the loss function with respect to the generator's output, poses a challenge. It requires access to the model parameters of the victim model θ_V. Since the victim model $\mathcal{M}_V$ is considered a black-box model from the attacker's perspective, we do not have direct access to θ_V. As a result, backpropagation cannot be applied to compute this derivative.

To overcome this limitation, we resort to an alternative approach that involves performing additional queries to approximate the gradient. Specifically, we employ zeroth-order gradient estimation techniques [25, 28] to estimate the gradient without explicitly accessing the victim model's parameters:

$$\hat{\nabla}_{\mathbf{G}} \mathcal{L}_{\mathcal{M}_G}(\mathbf{G}; u_i) = \frac{d \cdot \left(\mathcal{L}_{\mathcal{M}_G}(\mathbf{G} + \varepsilon u_i) - \mathcal{L}_{\mathcal{M}_G}(\mathbf{G}) \right)}{\varepsilon} u_i, \tag{9}$$

where u_i represents a random variable drawn from a d-dimensional unit sphere, with each component being uniformly distributed. Additionally, ε denotes a small positive constant. To estimate the gradient, we adopt an averaged version of the random gradient estimate, as proposed in previous works [6, 14, 19]. This involves computing the forward difference using m random directions, denoted as $\{u_1, u_2, ..u_m\}$. In our experiments, we set m to 2, utilizing two random directions to estimate the gradient. This choice is based on a trade-off between computational efficiency and gradient estimation accuracy. While increasing the value of m could potentially provide a more accurate gradient estimate, it also incurs a higher computational cost. Our experiments demonstrated that using two random directions already yielded sufficiently accurate gradient estimates for our specific application, striking a balance between accuracy and computational efficiency.

$$\hat{\nabla}_{\mathbf{G}} \mathcal{L}_{\mathcal{M}_G}(\mathbf{G}) = \frac{1}{m} \sum_{i=1}^{m} \hat{\nabla}_{\mathbf{G}} \mathcal{L}_{\mathcal{M}_G}(\mathbf{G}; u_i). \tag{10}$$

The expression $\hat{\nabla}_{\mathbf{G}} \mathcal{L}_{\mathcal{M}_G}$ represents an estimate of the true gradient $\nabla_{\mathbf{G}} \mathcal{L}_{\mathcal{M}_G}$. By substituting this gradient estimate into Equation 8, we obtain an approximation for the gradient of the generator's loss function, denoted as $\hat{\nabla}_{\theta_G} \mathcal{L}_{\mathcal{M}_G}$. This estimated gradient can be utilized to perform gradient descent, enabling the update of the generator model's parameters θ_G according to Equation 7. Through this parameter update, the generator model $\mathcal{M}_G$ is trained to generate synthetic graphs that facilitate the process of model extraction.

4.4.2 Type II Attack

As shown in Fig. 1(b), in this type of attack, the attacker solely relies on the gradient computed through surrogate $\mathcal{M}_S$. During the training process, the surrogate model learns to mimic the victim model, serving as an appropriate proxy to encourage the generator in generating diverse and informative graphs.

4.4.3 Type III Attack

Fig. 1(c) illustrates an overview of this attack type. When provided with a black-box Victim model, two surrogate models, surrogate 1 and surrogate 2, are generated. The final extracted model is an ensemble that consists of these two surrogate models. The model output is obtained through an equal-weight

soft voting mechanism, combining the predictions of the two surrogates. In accordance with a previous study by Rosenthal et al. [29], the generator training in this work incorporates disagreement losses to update the generator model. The disagreement loss measures the disparity between the predictions of a pair of surrogate models. In cases where two clones exhibit inconsistent predictions on specific samples, two scenarios arise: 1) If one of the clones aligns with the victim model's prediction, this sample assists the other clone in learning from the victim model. 2) If both clones' predictions do not match the victim model's prediction, they can both learn from this sample. Mathematically, the disagreement loss quantifies this difference by leveraging the standard deviation of the surrogate models' outputs for each corresponding node.

$$\mathcal{L}_{\mathcal{M}_G} = -\frac{1}{N}\sum_{i=1}^{N}\left[\mathrm{Std}\left(\mathbf{P}_{S1_i}, \mathbf{P}_{S2_i}\right)\right], \tag{11}$$

where Std denotes the standard deviation across the surrogate models. $\mathbf{P}_{S1_i}$ and $\mathbf{P}_{S2_i}$ represent the predictions of the two surrogate models, respectively. This loss function encourages the generator to produce data that prompts disagreement between two surrogate models when making predictions. The standard deviation quantifies this disagreement between the surrogates on generated data. By minimizing this loss, the training process aims to create data that exposes differences between the surrogates, allowing one of them to adapt and align better with the behavior of the victim model. This approach helps improve surrogate model accuracy by leveraging discrepancies in their predictions on generated data.

4.4.4 Training Procedure

Given the loss functions $\mathcal{L}_{\mathcal{M}_G}$ and $\mathcal{L}_{\mathcal{M}_S}$, similar to the training approach used in Generative Adversarial Networks (GAN) [10], we adopt an alternating training strategy for the generator and surrogate model, resembling a two-player game. The training process can be formulated as:

$$\min_{\mathcal{M}_S}\max_{\mathcal{M}_G}\mathbb{E}_{z\sim\mathcal{N}(0,1)}[\mathcal{D}(\mathcal{M}_V(\mathcal{M}_G(z)), \mathcal{M}_S(\mathcal{M}_G(z)))], \tag{12}$$

where $\mathcal{D}(\cdot)$ represents the discrepancy between the victim and surrogate models. The loss function used to update the generator $\mathcal{M}_G$ is identical to that of the surrogate model $\mathcal{M}_S$, with the exception that the objective is to maximize it.

We provide a concise summary of the StealGNN algorithm in Algorithm 1 (available in the appendix A.1). Our objective is to acquire a surrogate model $\mathcal{M}_S$ from a black-box victim model $\mathcal{M}_V$. Initially, both the surrogate model and generator are randomly initialized. During the generator optimization, the surrogate model parameters remain static, while the generator parameters θ_G are updated based on the chosen attack type. This procedure enables the generator to produce synthetic graphs for efficient knowledge extraction from the victim model. Conversely, in the surrogate training phase, the generator remains fixed, and the surrogate model parameters θ_S are updated. This training phase centers on replicating the victim model's behavior, ensuring the surrogate model captures the targeted knowledge. To maintain balanced training for both the generator and surrogate model, each training phase repeats n_G and n_S times, respectively, before advancing to the next epoch. This repetition enhances the learning and convergence of both models. Training iterations persist until the query budget Q is depleted, indicating that the attacker has exhausted the allotted number of queries for knowledge extraction from the victim model.

Table 2: Summary of datasets.

Dataset	#Nodes	#Edges	#Features	#Classes
Cora [31]	2,485	5,069	1,433	7
Pubmed [24]	19,717	44,324	500	3
A-Computers [32]	13,381	245,778	767	10
OGB-Arxiv [11]	169,343	1,166,243	128	40

5 Evaluation

In this section, we conduct experiments to evaluate the effectiveness of StealGNN. We present the experimental setup and provide evaluation results for node classification tasks, including both transductive and inductive scenarios. Additionally, we evaluate the performance of StealGNN on the link prediction task. Finally, we provide a detailed analysis of the model and parameters used in our experiments. For experimental results on the transductive scenario, please refer to the appendix A.3.

5.1 Experimental Setup

5.1.1 Datasets

To assess the effectiveness of our proposed framework, we conduct experiments using four publicly available benchmark datasets. The dataset statistics are summarized in Tab. 2. For Cora, Pubmed, and A-Computers, we follow the splitting strategy employed in previous work [32], where 20 nodes from each class are randomly sampled as labeled nodes, 30 nodes are used for validation, and the remaining nodes are assigned to the test set. For OGB-Arxiv, we adopt the random split configuration of 90K/29K/48K as established in prior research [12].

Using the same or similar data splits as previous research enables researchers to compare their results directly with existing literature. It ensures consistency in evaluation, making it easier to gauge the effectiveness of new methods or compare them with established approaches. We also explored alternative splits for these datasets, utilizing a 60%/20%/20% division for training, validation, and testing sets. This partitioning strategy allocated a larger proportion of nodes to the

training dataset, resulting in higher accuracy for the victim model when evaluated on the test set. Detailed results of these experiments can be found in the Tab. 5.

5.1.2 Victim Model ($\mathcal{M}_V$) and Surrogate Model ($\mathcal{M}_S$)

To evaluate the performance of the StealGNN framework, we employ widely used GNN models, namely GCN, GAT, and GraphSAGE. The victim model is also trained using the data partitioning scheme described above. Various architectures of the victim model, including different numbers of layers and hidden units, are trained and tested. The model with the highest test accuracy is then selected as the final victim model. Therefore, for different datasets and different models, their model architectures may vary. The architecture and accuracy of the victim model selected for the final experiment can be seen in Tab. 7 in the appendix A.2.

Experiments involving different types of surrogate models and architectures are presented in the Model Analysis and Parameter Analysis sections. In the main experiment, we have chosen to utilize a GCN model with 2 layers and 256 hidden units as the default surrogate model because it consistently performed well across different datasets, as shown in Fig. 4 and Fig. 5.

5.1.3 Generator ($\mathcal{M}_G$)

In our experiments, the choice of hyperparameters for the generator was motivated by their effectiveness in achieving our objectives. For the feature generator, we adopted a design inspired by [35], known for generating informative features. The hyperbolic tangent function was chosen as the final activation to ensure generated features fall within the range [-1, 1]. Regarding the structure generator, we used cosine similarity, which does not require additional training parameters to generate the adjacency matrix, simplifying the model. For the self-supervised training of the full parameter generator, a GCN model with 2 layers and 512 hidden units was selected to strike a balance between complexity and efficiency.

5.1.4 Implementation

For the node classification task, we perform comprehensive experiments on both inductive and transductive graph neural networks (GNNs). However, due to space limitations, the model experiments presented below default to the inductive scenario, where the learned GNN model is capable of generalizing to previously unseen graphs during training. Experiments conducted on transductive scenarios can be found in the appendix A.3. Our experiments are trained using the Adam optimizer [15]. During training, each query consists of the generator being trained $n_G = 2$ times with a learning rate of $1e-6$, while the surrogate model is trained $n_S = 5$ times with a learning rate of 0.001. The total number of queries Q is set to 100. By training the models for a maximum of 100 epochs, we ensure that the models have ample opportunities to converge and achieve satisfactory performance on our experimental dataset.

5.1.5 Evaluation Metric

In model extraction attacks, the objective is to closely align the behavior of the surrogate model with that of the victim model. To evaluate the performance of our attack, we employ two metrics: (1) Accuracy: We measure the accuracy of the attacker's surrogate model on the test dataset. The accuracy is computed as the ratio of correct predictions to the total number of predictions made by the surrogate model. This metric provides an assessment of the attack's effectiveness in replicating the victim model's behavior. (2) Fidelity: The fidelity metric quantifies the agreement between the predictions made by the surrogate model and the victim model. Fidelity serves as a complementary evaluation metric, capturing the similarity in prediction outcomes between the surrogate and target models. To mitigate the impact of randomness during training, we conduct multiple runs of each experiment with different random seeds. Specifically, we perform 5 runs for each experiment and report the average and standard deviation of both accuracy and fidelity across these runs. This approach allows us to capture the overall performance and the degree of variability in the results, providing a more robust evaluation of the attack method.

5.1.6 Baselines

StealGNN is the first study to address data-free model extraction specifically for GNNs. While there have been related methods in the past, they differ in important aspects. Firstly, methods like GNN model extraction or stealing [33, 39], although related, are not data-free techniques as they require access to some original data information. Secondly, there exist techniques such as GNN's data-free knowledge distillation [5, 44], but they are primarily designed for graph classification tasks and operate in white-box scenarios where gradient information of the teacher model is accessible. For each dataset, we conducted several experiments to compare the performance of our model:

- Real Data: We used real training data for model extraction. This allowed us to assess the impact of using artificial graphs for the extraction process.
- Random Graph: We generated random graphs using random noise without updating the parameters of the generator during the training process.
- Graph Structure Experiments: For each type of attack, we conducted experiments using three different graph structures:

Table 3: Experimental Results on the Induction Scenario for Node Classification Task. The table presents the mean and variance of 5 training runs. The brackets indicate the difference between the accuracy of the surrogate model and the accuracy of the victim model. Entries displayed in red font indicate that the surrogate model outperforms the victim model.

$\mathcal{M}_V$	Dataset	Cora		Pubmed		A-Computers		OGB-Arxiv	
	$\mathcal{M}_S$(GCN)	Accuracy	Fidelity	Accuracy	Fidelity	Accuracy	Fidelity	Accuracy	Fidelity
GAT	Real Data	80.09±0.89 (+0.53)	92.96±0.51	76.94±0.31(-0.12)	89.96±2.51	79.20±1.67 (-0.89)	90.51±1.12	53.89±1.02 (-0.77)	88.73±1.94
	Random Graph	68.26±3.84 (-11.30)	73.70±2.95	59.35±2.86 (-17.71)	71.79±2.49	55.63±4.61 (-24.46)	52.29±3.62	36.78±5.26 (-17.88)	63.52±4.39
	Attack I-$\mathbf{E}$	81.11±0.50 (+1.55)	93.09±0.26	**77.57±0.78** (+0.51)	92.09±2.56	65.68±1.91 (-14.41)	73.48±2.13	**52.17±2.94** (-2.49)	80.26±2.23
	Attack I-$\mathbf{A}_{cos}$	80.79±0.75 (+1.23)	93.90±0.42	77.15±0.93 (+0.09)	90.34±2.94	70.73±1.90 (-9.36)	81.18±1.18	51.72±2.31 (-2.94)	78.77±1.68
	Attack I-$\mathbf{A}_{fp}$	**81.23±0.68** (+1.67)	94.10±0.38	77.40±0.87 (+0.34)	90.90±2.76	**71.12±1.78** (-8.54)	81.45±1.10	52.15±2.23 (-2.51)	79.22±1.58
	Attack II-$\mathbf{E}$	80.98±0.57 (+1.42)	92.78±0.39	77.18±0.68 (+0.12)	91.14±1.34	68.76±1.98 (-11.33)	78.95±1.78	51.93±3.11 (-2.73)	79.34±2.58
	Attack II-$\mathbf{A}_{cos}$	81.17±0.71 (+1.61)	93.19±0.47	77.44±0.84 (+0.38)	91.60±2.82	71.78±1.38 (-8.31)	83.92±1.66	51.81±2.78 (-2.85)	79.01±2.08
	Attack II-$\mathbf{A}_{fp}$	**81.35±0.66** (+1.79)	93.50±0.43	**77.58±0.79** (+0.52)	91.88±2.78	**72.05±1.32** (-7.80)	84.10±1.62	**51.98±2.65** (-2.68)	78.87±1.99
	Attack III-$\mathbf{E}$	81.12±0.52 (+1.56)	93.02±0.36	77.30±0.64 (+0.24)	91.40±1.28	70.12±1.92 (-9.54)	79.12±1.62	51.78±2.88 (-2.63)	79.12±2.43
	Attack III-$\mathbf{A}_{cos}$	81.45±0.62 (+1.89)	93.80±0.41	77.65±0.74 (+0.59)	92.12±2.72	72.50±1.26 (-6.68)	84.35±1.56	52.05±2.58 (-2.61)	79.45±1.91
	Attack III-$\mathbf{A}_{fp}$	**81.68±0.58** (+2.12)	94.05±0.39	**77.80±0.70** (+0.76)	92.45±2.66	**73.20±1.18** (-5.46)	84.68±1.50	**52.32±2.50** (-2.34)	79.68±1.82
GCN	Real Data	81.09±0.69 (+0.49)	92.84±0.29	77.06±0.41 (+0.30)	97.86±0.49	79.50±0.66 (-1.13)	87.34±1.63	63.51±0.84 (-3.24)	85.93±0.92
	Random Graph	69.83±2.61 (-10.78)	81.79±2.28	62.78±2.49 (-13.98)	76.54±1.92	51.14±6.21 (-29.49)	63.85±3.28	49.73±9.37 (-17.02)	68.59±4.27
	Attack I-$\mathbf{E}$	80.67±0.45 (+0.06)	93.57±0.96	75.93±0.72 (-0.83)	89.57±1.29	68.94±2.12 (-11.69)	80.69±2.34	60.15±0.26 (-6.60)	82.29±1.36
	Attack I-$\mathbf{A}_{cos}$	80.68±0.29 (+0.07)	93.77±1.37	76.73±0.47 (-0.03)	95.82±1.03	69.55±2.75 (-11.08)	82.61±1.95	61.34±0.51 (-5.41)	80.76±1.26
	Attack I-$\mathbf{A}_{fp}$	**80.91±0.24** (+0.30)	93.96±1.24	**76.79±0.39** (+0.03)	95.98±0.81	**70.12±2.54** (-10.51)	82.83±1.76	**61.76±0.41** (-4.99)	80.96±1.02
	Attack II-$\mathbf{E}$	79.79±0.23 (-0.82)	92.81±0.79	74.18±1.57 (-2.58)	86.60±2.98	69.57±1.32 (-11.06)	83.01±1.48	60.37±1.39 (-6.38)	83.33±1.57
	Attack II-$\mathbf{A}_{cos}$	80.07±0.59 (-0.54)	92.84±0.31	76.47±0.80 (-0.29)	93.86±1.73	**70.09±1.49** (-10.54)	83.28±1.63	62.10±0.85 (-4.65)	81.50±0.93
	Attack II-$\mathbf{A}_{fp}$	**81.12±0.28** (+0.51)	93.65±0.12	**76.74±0.51** (+0.11)	94.97±0.23	69.59±0.92 (-11.04)	83.32±0.93	**62.59±0.45** (-4.16)	80.97±0.47
	Attack III-$\mathbf{E}$	80.78±0.31 (+0.17)	93.39±0.15	76.43±0.55 (-0.33)	94.51±0.37	70.11±1.05 (-10.52)	83.21±1.05	62.06±0.52 (-4.69)	81.16±0.55
	Attack III-$\mathbf{A}_{cos}$	81.05±0.25 (+0.44)	93.62±0.10	76.69±0.45 (-0.07)	94.83±0.18	69.79±0.83 (-10.84)	83.42±0.89	62.46±0.40 (-4.29)	80.97±0.41
	Attack III-$\mathbf{A}_{fp}$	**81.28±0.21** (+0.67)	93.79±0.08	**76.85±0.38** (+0.22)	95.05±0.11	**70.21±0.66** (-10.42)	83.64±0.68	**62.87±0.32** (-3.88)	80.81±0.35
SAGE	Real Data	79.84±1.25 (+0.51)	92.57±0.28	76.76±0.20 (-0.47)	92.64±0.77	79.61±1.59 (+0.37)	88.72±0.95	65.35±0.79 (-3.83)	85.72±1.45
	Random Graph	60.39±2.95 (-18.94)	78.39±3.38	58.75±3.38 (-18.48)	80.28±2.96	52.75±4.67 (-26.49)	68.39±4.21	49.89±3.82 (-19.29)	69.93±2.83
	Attack I-$\mathbf{E}$	**81.32±0.65** (+1.99)	93.01±0.12	77.09±0.17 (-0.14)	92.97±0.44	65.39±2.49 (-13.85)	76.44±1.36	**58.22±3.85** (-10.96)	78.36±2.03
	Attack I-$\mathbf{A}_{cos}$	80.89±0.47 (+1.56)	93.97±0.49	77.24±0.57 (+0.01)	94.74±0.96	**69.88±2.22** (-9.36)	83.26±1.27	57.46±3.79 (-11.72)	74.29±2.57
	Attack I-$\mathbf{A}_{fp}$	81.10±0.35 (+1.77)	93.85±0.33	**77.40±0.42** (+0.17)	94.52±0.82	69.76±1.93 (-8.42)	83.48±1.12	57.76±3.45 (-11.48)	74.15±2.24
	Attack II-$\mathbf{E}$	**81.28±0.67** (+1.95)	93.92±0.70	76.88±0.32 (-0.35)	93.11±1.40	67.39±3.08 (-11.85)	78.92±2.27	57.15±3.59 (-12.03)	73.38±1.93
	Attack II-$\mathbf{A}_{cos}$	81.01±0.51 (+1.68)	95.14±0.17	77.13±0.40(-0.10)	93.54±1.84	**70.17±1.89** (-9.07)	84.06±1.95	57.69±3.51 (-11.49)	75.59±1.73
	Attack II-$\mathbf{A}_{fp}$	80.98±0.41 (+1.65)	93.68±0.39	**77.21±0.48** (-0.02)	94.38±0.89	69.99±2.08 (-9.25)	83.32±1.18	**57.91±3.62** (-11.27)	73.94±2.48
	Attack III-$\mathbf{E}$	**81.56±0.45** (+2.23)	93.28±0.09	77.22±0.13 (-0.01)	93.36±0.29	69.02±2.23 (-10.22)	81.78±1.14	**58.61±3.56** (-10.57)	78.74±1.73
	Attack III-$\mathbf{A}_{cos}$	81.45±0.63 (+2.12)	93.10±0.10	77.20±0.15 (-0.03)	92.80±0.42	69.50±2.46 (-9.74)	83.60±1.30	58.00±3.75 (-11.18)	78.20±2.00
	Attack III-$\mathbf{A}_{fp}$	81.55±0.61 (+2.22)	93.15±0.09	**77.25±0.14** (+0.02)	92.90±0.41	**70.55±2.43** (-8.69)	86.65±1.32	58.55±3.78 (-10.63)	78.25±1.97

(a) Self-Loop Graph Structure ($\mathbf{E}$): In this case, the graph structure consisted of only self-loops, and the adjacency matrix was represented by matrix $\mathbf{E}$.

(b) Cosine Similarity Graph Structure ($\mathbf{A}_{cos}$): The graph structure was generated based on cosine similarity, and the adjacency matrix was represented by matrix $\mathbf{A}_{cos}$.

(c) Full Parameterization Graph Structure ($\mathbf{A}_{fp}$): The graph structure was generated using a parametric approach, and the adjacency matrix was represented by matrix $\mathbf{A}_{fp}$.

By conducting these experiments with different graph structures and utilizing both real data and random graphs, we aimed to evaluate the effectiveness and performance of our model extraction approach.

5.2 Performance Evaluation

Based on the node classification results presented in Tab. 3 and the link prediction results in Tab. 4, it can be observed that all three types of model extraction attacks yield promising results. The Real Data approach, which utilizes real training datasets for extraction, demonstrates a high level of effectiveness. This method resembles traditional knowledge distillation techniques, resulting in a surrogate model that closely approximates the behavior of the victim model. The surrogate model's performance is very similar to that of the victim model, indicating successful extraction of knowledge. On the other hand, the results obtained from the Random Graph approach are noticeably inferior. This can be attributed to the randomly generated graph, which lacks any form of training. Consequently, the distribution of the real training data significantly differs from that of the randomly generated graph,

Table 4: Experimental Results on Link Prediction Tasks. The table displays the accuracy results of the experiments. Note that the OGB-Arxiv dataset is excluded from the analysis due to OOM issues.

$\mathcal{M}_V$	Dataset	Cora	Pubmed	A-computers
GAT	Real Data	89.30±0.59 (-0.38)	65.70±1.58 (-4.67)	84.64±1.78 (+1.89)
	Random Graph	75.32±2.12 (-14.36)	50.21±3.89 (-20.16)	66.79±2.78 (-15.96)
	Attack I- $\hat{\mathbf{E}}$	89.01±0.56 (-0.67)	64.78±1.61 (-5.59)	80.29±1.48 (-2.46)
	Attack I- $\mathbf{A}_{cos}$	88.71±0.32 (-0.97)	**64.98±1.92** (-5.39)	**80.62±1.19** (-2.13)
	Attack I- $\mathbf{A}_{fp}$	**89.12±0.22** (-0.56)	64.25±1.38 (-6.12)	79.92±0.92 (-2.83)
	Attack II- $\hat{\mathbf{E}}$	88.96±0.63 (-0.72)	64.58±1.82 (-5.79)	**80.19±1.19** (-2.56)
	Attack II- $\mathbf{A}_{cos}$	**89.38±0.47** (-0.30)	64.39±1.25 (-5.98)	79.89±0.92 (-2.86)
	Attack II- $\mathbf{A}_{fp}$	88.95±0.62 (-0.73)	**66.12±1.61** (-4.25)	78.98±1.28 (-3.77)
	Attack III- $\hat{\mathbf{E}}$	88.95±0.50 (-0.73)	**66.78±1.32** (-3.59)	79.47±0.98 (-3.28)
	Attack III- $\mathbf{A}_{cos}$	89.25±0.53 (-0.43)	65.78±1.42 (-4.59)	79.42±0.95 (-3.33)
	Attack III- $\mathbf{A}_{fp}$	**89.45±0.35** (-0.23)	66.25±1.15 (-4.12)	**80.02±0.86** (-2.73)
GCN	Real data	89.59±0.23 (-0.09)	87.72±0.18 (-0.38)	90.70±0.48 (-3.14)
	Random Graph	80.12±2.18 (-9.56)	72.85±3.29 (-15.25)	80.68±4.56 (-13.16)
	Attack I- $\hat{\mathbf{E}}$	90.78±0.73 (+1.10)	85.54±1.09 (-2.56)	85.49±1.39 (-8.35)
	Attack I- $\mathbf{A}_{cos}$	**90.83±0.41** (+1.15)	84.20±1.67 (-2.90)	84.54±1.10 (-9.30)
	Attack I- $\mathbf{A}_{fp}$	89.70±0.60 (-0.02)	**87.45±0.85** (-0.76)	**86.55±1.42** (-7.29)
	Attack II- $\hat{\mathbf{E}}$	**89.95±0.52** (+0.27)	87.69±0.74 (-0.41)	**86.95±1.29** (-6.89)
	Attack II- $\mathbf{A}_{cos}$	89.09±0.25 (-0.59)	88.05±0.28 (-0.05)	85.19±1.41 (-8.65)
	Attack II- $\mathbf{A}_{fp}$	89.30±0.22 (-0.38)	**88.25±0.25** (+0.15)	85.50±1.28 (-8.34)
	Attack III- $\hat{\mathbf{E}}$	90.25±0.45 (+0.57)	87.85±0.62 (-0.25)	87.15±1.05 (-5.69)
	Attack III- $\mathbf{A}_{cos}$	90.40±0.42 (+0.72)	88.09±0.55 (-0.01)	87.35±0.98 (-5.49)
	Attack III- $\mathbf{A}_{fp}$	**90.65±0.40** (+0.97)	**88.12±0.52** (+0.02)	**87.48±0.98** (-5.36)
SAGE	Real data	91.15±0.20 (-1.32)	87.90±0.09 (+1.06)	90.38±1.64 (+4.08)
	Random Graph	77.25±1.35 (-10.54)	75.60±1.75 (-11.34)	71.10±0.92 (-11.69)
	Attack I- $\hat{\mathbf{E}}$	**88.68±0.54** (-3.79)	85.79±0.48 (-1.05)	83.07±1.26 (-3.23)
	Attack I- $\mathbf{A}_{cos}$	87.79±0.96 (-4.68)	86.94±0.51 (+0.00)	82.79±1.57 (-3.51)
	Attack I- $\mathbf{A}_{fp}$	88.05±0.45 (-4.42)	**87.30±0.55** (+0.36)	**84.10±1.20** (-2.20)
	Attack II- $\hat{\mathbf{E}}$	89.01±0.56 (-3.46)	86.77±0.39 (-0.07)	**82.45±1.38** (-3.85)
	Attack II- $\mathbf{A}_{cos}$	**90.04±0.47** (-2.43)	**88.26±0.08** (+1.42)	81.25±1.49 (-5.05)
	Attack II- $\mathbf{A}_{fp}$	88.50±0.35 (-4.97)	87.80±0.50 (+0.96)	82.28±0.90 (-4.02)
	Attack III- $\hat{\mathbf{E}}$	90.05±0.50 (-2.42)	87.32±0.30 (+0.48)	83.53±1.30 (-2.77)
	Attack III- $\mathbf{A}_{cos}$	90.31±0.40 (-2.16)	**88.15±0.10** (+1.31)	83.93±1.30 (-2.37)
	Attack III- $\mathbf{A}_{fp}$	**90.59±0.40** (-1.88)	87.43±0.25 (+0.59)	**84.78±1.20** (-1.52)

making it challenging to extract knowledge from the victim model into the surrogate model. As a result, the surrogate model's performance suffers.

The experimental results demonstrate that all three types of attacks have achieved notable performance. Attack Type I and Attack Type II exhibit comparable results, while Attack Type III outperforms the other two and consistently achieves the highest accuracy in several datasets. Attack Type III leverages the prediction inconsistency between the two surrogate models to train the generator. This approach ensures that the training process continuously generates informative graphs, facilitating the surrogate model's improved learning from the victim model. As a result, Attack Type 3 exhibits superior performance, achieving optimal accuracy in multiple experimental settings.

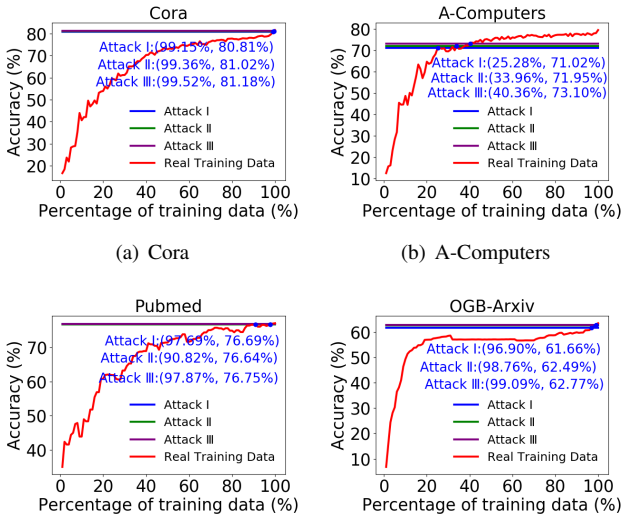
Among the three graph structure generation methods, the full-parameter approach stands out as the most effective. It excels due to its capability to generate a diverse range of graph structures through self-supervised training. With a full-parameter model, the generator has access to a larger parameter set, allowing it to capture a broader spectrum of graph structures during training. This flexibility enables the generator to produce diverse graph structures, contributing to more effective knowledge transfer from the victim model to the surrogate models. In contrast, the Self-Loop and cosine similarity methods may be limited in generating diverse graph structures. Self-Loop lacks a specific training process, and cosine similarity structure generation doesn't require training parameters, potentially restricting their ability to generate a wide range of graph structures compared to the full-parameter method. The superior performance of the full-parameter approach underscores the significance of diverse graph structures in facilitating effective knowledge transfer during model extraction attacks.

Table 5: Experimental Results for a 60%/20%/20% Data Splitting Scheme. Values within parentheses indicate improvements compared to another data partitioning scheme. $\mathcal{M}_S$ is also GCN.

Dataset	Cora	Pubmed	A-Computers	OGB-Arxiv
$\mathcal{M}_V$ (GCN)	88.72±0.32 (+8.11)	89.59±0.16 (+12.83)	85.46±0.64 (+4.83)	69.58±0.48 (+2.83)
$\mathcal{M}_S$ (Attack I-$\mathbf{A}_{fp}$)	87.55±0.61 (+6.64)	87.98±0.73 (+11.19)	77.51±1.11 (+7.39)	64.52±0.75 (+2.76)
$\mathcal{M}_S$ (Attack II-$\mathbf{A}_{fp}$)	87.84±0.73 (+6.72)	88.06±0.82 (+11.32)	77.89±1.52 (+8.30)	65.92±0.64 (+3.33)
$\mathcal{M}_S$ (Attack III-$\mathbf{A}_{fp}$)	88.14±0.65 (+6.86)	88.72±0.67 (+11.87)	78.35±1.25 (+8.14)	67.33±0.82 (+4.46)

Tab. 5 presents the results of using an alternative data partitioning scheme. Employing a 60%/20%/20% data split increases the number of nodes in the training dataset, resulting in improved accuracy for the victim model. Correspondingly, the surrogate model, which extracts model information from a better victim model, also exhibits enhanced accuracy. This underscores the effectiveness of STEALGNN - as the victim model's accuracy improves, it imparts richer knowledge to the surrogate model, showcasing the knowledge transfer capabilities of the framework.

The experimental results reveal a noteworthy trend where the surrogate model's accuracy often exceeds that of Real Data and occasionally surpasses the victim model's performance (highlighted in red). One plausible explanation, supported by prior research [33], revolves around the presence of noise in real training data. Real-world data can inherently contain noise, including inconsistencies and inaccuracies. To mitigate this, various data denoising techniques have been proposed [42]. In our approach, a unique feature is the use of a graph generator explicitly trained for model extraction. This generator produces synthetic data tailored for the extraction process, resulting in cleaner and more suitable training data for surrogate models. As a result, we obtain surrogate models that not only replicate the victim model's behavior but, in some cases, even outperform it.



(a) Cora (b) A-Computers

(c) Pubmed (d) OGB-Arxiv

Figure 3: Training with Different Percentages of Real Data. The red curve represents the results as the percentage of real data used for model extraction gradually increases.The intersection point in the graph indicates the required rate of real training data to achieve the same performance as our data-free Attack.

5.3 Model Analysis

In order to gain a deeper understanding of the effectiveness of our data-free attack model, we conducted further analysis to evaluate the role of the generated graphs. Specifically, we compared the performance of the generated data with varying percentages of real data to quantify its impact. Furthermore, we also examined how the structure of the victim model and the surrogate model impacts the performance of the attack. By varying the architecture and parameters of these models, we can assess their influence on the effectiveness of the extraction attack. This analysis provides insights into the sensitivity of the attack to different model structures and can help identify optimal configurations for achieving higher performance.

5.3.1 Comparison with Existing Methods

Directly comparing with existing GNN model extraction methods is challenging due to differing setups. They assume access to original node features or structures, while ours are generated. We held StealGNN's node features or adjacency matrix fixed as real data, while the other was generated (w/ A w/o X: real adjacency matrix, generated node features). From Tab. 6, it is evident that our model consistently outperforms existing works. This highlights that our framework not only addresses scenarios with no access to any real data but also exhibits superior performance in scenarios where partial real data information is available.

Table 6: Comparison between the work by Wu et al. [39] and StealGNN. "w/ A w/o X" denotes that the adjacency matrix of the generated graph is based on real data, while node features are synthetic. Conversely, "w/ X w/o A" indicates that node features are real data, while the adjacency matrix is synthetic.

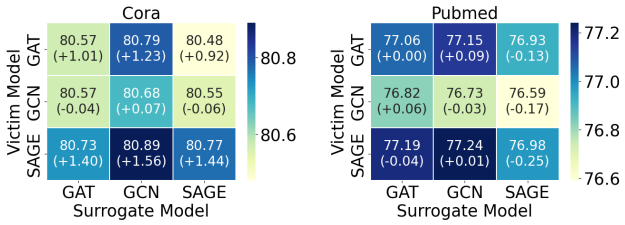
	w/ A w/o X		w/ X w/o A	
Dataset	Work [39]	Attack III-$\mathbf{A}_{fp}$	Work [39]	Attack III-$\mathbf{A}_{fp}$
Cora	79.80±0.52	80.49±0.34	79.29±0.46	80.95±0.57
Pubmed	73.65±0.86	77.03±0.62	75.19±0.38	76.86±0.51
A-Computers	70.88±0.73	74.52±0.79	69.73±0.82	72.37±0.55
OGB-Arxiv	56.35±0.59	62.28±0.68	57.49±0.83	61.71±0.47

5.3.2 Quantitative Analysis: Role of the Generated Graph

We conduct a detailed analysis of the impact of both the generated graphs and the real data on the performance of the surrogate model. To perform this analysis, we vary the percentages of real data from 0% to 100% and observe the corresponding attack results presented in Fig. 3. In the figure, the x-axis represents the percentage of real training data required, while the y-axis denotes the accuracy rate. The intersection point between the attack's straight line and the curve representing the real data corresponds to the percentage of real data needed to attain our performance. The results reveal an interesting finding: a substantial proportion of real data is required to achieve accuracy levels comparable to our data-free attacks. This implies that the generated graphs play a crucial role in the knowledge transfer process. As the percentage of real data decreases, the attack accuracy decreases as well, indicating that the generated graphs become increasingly essential in compensating for the lack of real data. They demonstrate that the generated graphs effectively capture and transfer valuable information from the victim model, enabling the surrogate model to achieve high accuracy even in the absence of real data.

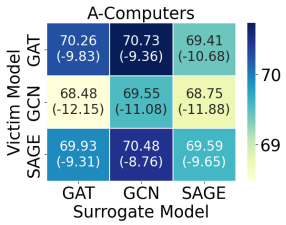
5.3.3 Impact of Different Combinations of Surrogate and Victim Models

In real-world scenarios, attackers do not have access to the architecture of the victim models. To assess the reliability of our attacker in extracting victim models under such circumstances, we conducted experiments using different combinations of surrogate and victim models for Attack I-$\mathbf{A}_{cos}$. With our experimental setup, which includes three victim models and three surrogate models, we explored nine different combinations for each dataset. Fig. 4 demonstrates that our attacker is capable of constructing viable surrogate models across various combinations. This observation confirms the effectiveness of our attack in successfully extracting useful knowledge from victim models, regardless of the specific combination of surrogate and victim models.

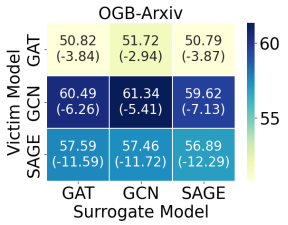


(a) Cora

(b) Pubmed

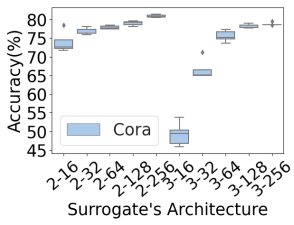


(c) A-Computers

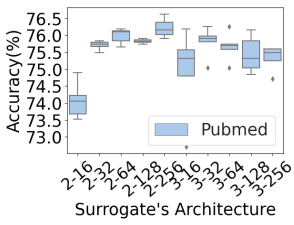


(d) OGB-Arxiv

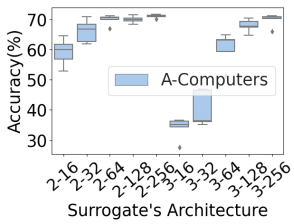
Figure 4: Performance of Different Victim Models with Different Surrogate Models.



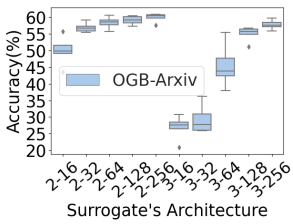
(a) Cora



(b) Pubmed

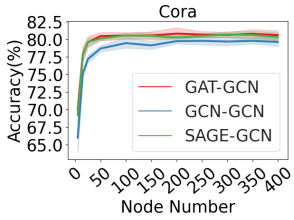


(c) A-Computers

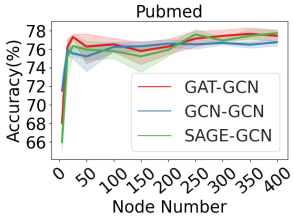


(d) OGB-Arxiv

Figure 5: Impact of Different Surrogate Architectures: Number of Layers and Hidden Units in the Surrogate Model.



(a) Cora



(b) Pubmed

Figure 6: Impact of Generated Graph Size: Number of Nodes in Generated Graph. The legend means (Victim-surrogate)

5.3.4 Impact of Different Surrogate Model Structures

To assess the robustness of our attack in different scenarios, we perform ablation experiments to examine the influence of different surrogate model architectures on the attack's effectiveness. Fig. 4 demonstrates that using GCN as a surrogate model consistently leads to more reliable model extraction. We further investigate the impact of various GCN architectures on the performance of our attack I-$\mathbf{A}_{cos}$. From the observations in Fig. 5, we draw the following conclusions: (1) Accuracy tends to be higher when the surrogate model employs a two-layer structure with 256 hidden units. (2) In contrast, the effect is significantly poorer when the number of hidden units is only 16. These findings indicate that the number of hidden units in the surrogatemodel has a substantial impact on the success of model extraction. Surrogate models with a higher number of parameters are more effective in extracting the unknown architecture of the victim model.

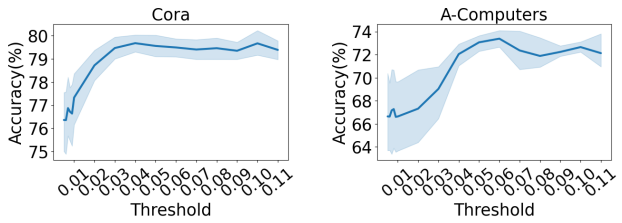
5.4 Parameter Analysis

5.4.1 Impact of the Size of Generated Graph

We investigate how the size of the generated graph, specifically the number of nodes, affects the performance of our attack. We conduct experiments on the Cora and Pubmed datasets, varying the node number of the generated graph from 5 to 400, as shown in Fig. 6: (1) Even with a generated graph containing only 50 nodes, our data-free model extraction attacks still achieve satisfactory accuracy. This highlights the effectiveness of our approach, as even a relatively small graph can capture and transfer valuable model information. (2) In Fig. 6 (b), we observe that the attack's performance tends to improve as the number of nodes in the generated graph increases. This suggests that a larger generated graph is more effective in capturing the complex and rich information of the victim model. In all of our experiments, we set the number of generated nodes to 250, which yields favorable results.

5.4.2 Impact of the Threshold τ

The threshold τ of the generated structures plays a crucial role in controlling the sparsity of the adjacency matrix. When the threshold exceeds 0.11, the model extraction performance significantly decreases due to the sparse nature of the generated graph, where there are almost no edges present. To understand the impact of threshold τ on the attack performance, we conducted experiments by varying τ from 0.005 to 0.11 on the Cora and A-Computers datasets. The results are shown in Fig. 7. From the figure, we observe that smaller thresholds, corresponding to denser graphs, lead to lower accuracy in the attack. This can be attributed to the fact that overly dense graph structures introduce significant disparities between the distribution of real data and generated data,



(a) Cora (b) A-Computers

Figure 7: Impact of Threshold τ.

hindering the learning process of the surrogate model. In our experiments, we set τ to 0.1 as it achieved satisfactory results.

6 Discussion

Limitation. An inherent limitation is its reliance on query access to the target GNN model. In practical scenarios, unrestricted query access may not always be feasible due to factors like access restrictions, cost considerations, or privacy concerns. These limitations can hinder the attacker's ability to execute a sufficient number of queries. Additionally, query-based attacks may not be suitable for all applications, where the cost or impact of querying the target model might be prohibitive. Future research should explore alternative attack strategies that reduce reliance on query access, thereby expanding the practical scope of GNN model extraction attacks and addressing scenarios with limited query access.

Defense. Prior defense strategies against model extraction attacks have often focused on adding noise to the output probabilities while maintaining the label outputs, effectively perturbing the probability distribution while preserving the top-1 label [16, 27]. However, it is crucial to note that this type of defense approach prove ineffective against StealGNN due to its inherent hard-label setting. In StealGNN, the attacker solely relies on the data's label information for model extraction, making it impervious to perturbations in output probabilities that do not alter the top-1 label. Therefore, traditional noise injection defenses that preserve label accuracy might not deter StealGNN effectively. Moreover, employing a defense strategy that deliberately introduces incorrect labels to the victim model's output can be detrimental to its performance. While this approach may disrupt model extraction attempts, it comes at the cost of degrading the accuracy and reliability of the GNN model's predictions, which is undesirable in practical applications.

To address the vulnerability of GNN models to StealGNN and similar model extraction attacks, it is imperative to explore alternative countermeasures that strike a balance between security and model performance. Some potential countermeasures and research directions include: (1) Adaptive Defenses: Developing adaptive defense mechanisms that dynamically adjust the model's behavior in response to potential model extraction threats. Such defenses could selectively introduce perturbations or noise when they detect suspicious querying patterns while maintaining performance in normal use cases. (2) Privacy-Preserving Techniques: Incorporating privacy-preserving techniques, such as differential privacy, into GNN models can add an additional layer of protection against model extraction attacks. These techniques can help obscure sensitive information in model responses.

7 Related Work

Model extraction attacks against GNNs. Graph learning models are indeed vulnerable to model extraction attacks, which aim to construct a surrogate model that closely matches the performance or prediction distributions of the target model. Our proposed extraction attack approach differs from existing methods in the context of Graph Neural Networks (GNNs). Prior model extraction attacks against GNNs, such as the work by Defazio et al. [4] and Wu et al. [39], typically assume that the attacker has access to information about the victim model's original training data, including node features, graph structure, or subgraphs. These attacks involve perturbing subgraphs or reconstructing missing attributes/edges based on auxiliary information to train a surrogate model that mimics the target model's behavior. These approaches primarily focus on transductive GNNs, and the problem of predicting unseen nodes has not been extensively explored. While Shen et al. [33] consider model stealing attacks against inductive GNNs and perform security risk assessments, their assumption of having access to node features from the same graph distribution used to train the victim model is also unrealistic in many real-world scenarios.

8 Conclusion

In this study, we explored data-free model extraction attacks on GNNs to assess their vulnerability without access to original training data. Through extensive experiments and analysis, we uncovered the effectiveness and limitations of these attacks. Our findings reveal that data-free model extraction can achieve high success rates in extracting knowledge from victim GNN models. These attacks employ synthetic graph generation and surrogate models to approximate the victim model, resulting in accurate proxies that capture its behavior.

Acknowledgments

We thank the anonymous reviewers and our shepherd for constructive feedback. This work is supported by the National Natural Science Foundation of China (No. U20B2045, U22B2038, 62192784, 62172052, 62002029) and the Key Laboratory of Trustworthy Distributed Computing and Service (BUPT), Ministry of Education

References

[1] Yihan Cao, Siyu Li, Yixin Liu, Zhiling Yan, Yutong Dai, Philip S Yu, and Lichao Sun. A comprehensive survey of ai-generated content (aigc): A history of generative ai from gan to chatgpt. *arXiv preprint arXiv:2303.04226*, 2023.

[2] Yu Chen, Lingfei Wu, and Mohammed Zaki. Iterative deep graph learning for graph neural networks: Better and robust node embeddings. *Advances in neural information processing systems*, 33:19314–19326, 2020.

[3] Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

[4] David DeFazio and Arti Ramesh. Adversarial model extraction on graph neural networks. *arXiv preprint arXiv:1912.07721*, 2019.

[5] Xiang Deng and Zhongfei Zhang. Graph-free knowledge distillation for graph neural networks. *arXiv preprint arXiv:2105.07519*, 2021.

[6] John C Duchi, Michael I Jordan, Martin J Wainwright, and Andre Wibisono. Optimal rates for zero-order convex optimization: The power of two function evaluations. *IEEE Transactions on Information Theory*, 61(5):2788–2806, 2015.

[7] Bahare Fatemi, Layla El Asri, and Seyed Mehran Kazemi. Slaps: Self-supervision improves structure learning for graph neural networks. *Advances in Neural Information Processing Systems*, 34:22667–22681, 2021.

[8] Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He. Learning discrete structures for graph neural networks. In *International conference on machine learning*, pages 1972–1982. PMLR, 2019.

[9] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[10] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *NeurIPS*, 27, 2014.

[11] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *ArXiv*, abs/2005.00687, 2020.

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

[13] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. High accuracy and high fidelity extraction of neural networks. In *29th USENIX security symposium (USENIX Security 20)*, pages 1345–1362, 2020.

[14] Sanjay Kariyappa, Atul Prakash, and Moinuddin K Qureshi. Maze: Data-free model stealing attack using zeroth-order gradient estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13814–13823, 2021.

[15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[16] Taesung Lee, Benjamin Edwards, Ian Molloy, and Dong Su. Defending against neural network model stealing attacks using deceptive perturbations. In *2019 IEEE Security and Privacy Workshops (SPW)*, pages 43–49. IEEE, 2019.

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

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

[19] Sijia Liu, Jie Chen, Pin-Yu Chen, and Alfred Hero. Zeroth-order online alternating direction method of multipliers: Convergence analysis and applications. In *International Conference on Artificial Intelligence and Statistics*, pages 288–297. PMLR, 2018.

[20] Yixin Liu, Chenrui Fan, Pan Zhou, and Lichao Sun. Unlearnable graph: Protecting graphs from unauthorized exploitation. *arXiv preprint arXiv:2303.02568*, 2023.

[21] Yuang Liu, Wei Zhang, Jun Wang, and Jianyong Wang. Data-free knowledge transfer: A survey. *arXiv preprint arXiv:2112.15278*, 2021.

[22] Yahui Long, Min Wu, Yong Liu, Yuan Fang, Chee Keong Kwoh, Jinmiao Chen, Jiawei Luo, and Xiaoli Li. Pre-training graph neural networks for link prediction in biomedical networks. *Bioinformatics*, 38(8):2254–2262, 2022.

[23] Rafael Müller, Simon Kornblith, and Geoffrey E Hinton. When does label smoothing help? *Advances in neural information processing systems*, 32, 2019.

[24] Galileo Namata, Ben London, Lise Getoor, Bert Huang, and U Edu. Query-driven active surveying for collective classification. In *10th International Workshop on Mining and Learning with Graphs*, volume 8, page 1, 2012.

[25] Yurii Nesterov and Vladimir Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2):527–566, 2017.

[26] Xichuan Niu, Bofang Li, Chenliang Li, Rong Xiao, Haochuan Sun, Hongbo Deng, and Zhenzhong Chen. A dual heterogeneous graph attention network to improve long-tail performance for shop search in e-commerce. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3405–3415, 2020.

[27] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Knockoff nets: Stealing functionality of black-box models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4954–4963, 2019.

[28] Boris T Polyak. Introduction to optimization. optimization software. *Inc., Publications Division, New York*, 1:32, 1987.

[29] Jonathan Rosenthal, Eric Enouen, Hung Viet Pham, and Lin Tan. Disguide: Disagreement-guided data-free model extraction. 2023.

[30] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

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

[32] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

[33] Yun Shen, Xinlei He, Yufei Han, and Yang Zhang. Model stealing attacks against inductive graph neural networks. *arXiv preprint arXiv:2112.08331*, 2021.

[34] Lichao Sun, Yingtong Dou, Carl Yang, Kai Zhang, Ji Wang, S Yu Philip, Lifang He, and Bo Li. Adversarial attack and defense on graph data: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[35] Jean-Baptiste Truong, Pratyush Maini, Robert J Walls, and Nicolas Papernot. Data-free model extraction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4771–4780, 2021.

[36] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.

[37] Jianian Wang, Sheng Zhang, Yanghua Xiao, and Rui Song. A review on graph neural network methods in financial applications. *arXiv preprint arXiv:2111.15367*, 2021.

[38] Yu Wang, Lifu Huang, Philip S Yu, and Lichao Sun. Membership inference attacks on knowledge graphs. *arXiv preprint arXiv:2104.08273*, 2021.

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

[40] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[41] Carl Yang, Haonan Wang, Ke Zhang, Liang Chen, and Lichao Sun. Secure deep graph generation with link differential privacy. *arXiv preprint arXiv:2005.00455*, 2020.

[42] Tong Zhao, Wei Jin, Yozen Liu, Yingheng Wang, Gang Liu, Stephan Günnemann, Neil Shah, and Meng Jiang. Graph data augmentation for graph machine learning: A survey. *arXiv preprint arXiv:2202.08871*, 2022.

[43] Ce Zhou, Qian Li, Chen Li, Jun Yu, Yixin Liu, Guangjing Wang, Kai Zhang, Cheng Ji, Qiben Yan, Lifang He, et al. A comprehensive survey on pretrained foundation models: A history from bert to chatgpt. *arXiv preprint arXiv:2302.09419*, 2023.

[44] Yuanxin Zhuang, Lingjuan Lyu, Chuan Shi, Carl Yang, and Lichao Sun. Data-free adversarial knowledge distillation for graph neural networks. *arXiv preprint arXiv:2205.03811*, 2022.

Algorithm 1 STEALGNN

Input: Query budget Q, victim model $\mathcal{M}_V(\mathbf{G};\theta_V)$, generator iterations n_G, surrogate iterations n_S, generator learning rate η_1, surrogate learning rate η_2.
Output: A comparable surrogate model $\mathcal{M}_S(\mathbf{G};\theta_S)$.

Randomly initialize $\mathcal{M}_S(\mathbf{G};\theta_S)$ and $\mathcal{M}_G(z;\theta_G)$;
for $i = 1, \cdots, Q$ **do**
  //Training Generator
  for $j = 1 \cdots n_G$ **do**
    Generate graph $\mathbf{G}$ from z with $\mathcal{M}_G(z;\theta_G)$;
    if launch Type I Attack **then**
      Approximate gradient $\nabla_{\theta_G}\mathcal{L}_{\mathcal{M}_G}$;
    else if launch Type II Attack **then**
      Compute $\nabla_{\theta_G}\mathcal{L}_{\mathcal{M}_G}$ only through $\mathcal{M}_S$;
    end if
    $\theta_{\mathbf{G}} = \theta_{\mathbf{G}} - \eta_1 \nabla_{\theta_G}\mathcal{L}_{\mathcal{M}_G}$;
    if launch Type III Attack **then**
      Compute $\nabla_{\theta_G}\mathcal{L}_{\text{Std}}$ through $\mathcal{M}_{S1}$ and $\mathcal{M}_{S2}$;
      $\theta_{\mathbf{G}} = \theta_{\mathbf{G}} - \eta_1 \nabla_{\theta_G}\mathcal{L}_{\text{Std}}$;
    end if
  end for
  // Training Surrogate Model
  for $j = 1 \cdots n_S$ **do**
    Generate graph $\mathbf{G}$ from z with $\mathcal{M}_G(z;\theta_G)$;
    Calculate gradient $\nabla_{\theta_S}\mathcal{L}_{\mathcal{M}_S}$;
    $\theta_{\mathbf{S}} = \theta_{\mathbf{S}} - \eta_2 \nabla_{\theta_S}\mathcal{L}_{\mathcal{M}_S}$;
  end for
end for

A Appendix

A.1 Algorithm

Algorithm 1 iteratively improves the surrogate model's performance by leveraging the generated synthetic data and optimizing both the generator and surrogate model parameters.

A.2 Victim Model Selection

Our framework is designed to extract knowledge from message-passing-based GNN models. In our experiments, we consider three representative GNN models: GCN, GAT, and GraphSAGE, which serve as the victim models. To select the most suitable pre-trained victim model, we conduct experiments with various architectures for each of these three models. We explore different combinations of the number of layers and hidden units, ranging from 1 to 5 and 16 to 256, respectively. The goal is to find the architecture that achieves the highest accuracy. After thorough experimentation, we determine the final architecture and accuracy of the chosen victim models, which are presented in Tab. 7. These models serve as the targets for our data-free model extraction attacks.

A.3 Tranductive Node Classification

Tab. 8 presents the results of node classification experiments in transductive scenarios. The table compares the performance of the three attacks using Real data and Random Graph. The results demonstrate that our framework achieves excellent performance even in the transductive scenario. This finding further confirms the versatility of our model, which can be successfully applied in various scenarios, including both transductive and inductive settings for node classification and link prediction tasks.

A.4 Connection with Real-World Cases

Type I Attack: Knowledge Transfer through Estimation:
(1) Real-World Scenario: Imagine a scenario in which a company has developed a highly effective recommendation system using a proprietary graph neural network for personalized content recommendations to users. Competing companies may want to extract knowledge from this system without direct access to the model's parameters or training data.
(2) Application: In this case, attackers can use a surrogate model to estimate gradients of the victim model's behavior by querying it with various inputs (e.g., user profiles, content features). They use these gradients to improve their own recommendation system, effectively transferring the valuable knowledge from the victim model.

Type II Attack: Gradual Alignment with Surrogate Model:
(1) Real-World Scenario: Consider a situation where a pharmaceutical company has developed a GNN-based model to predict the effectiveness of drug compounds for specific diseases. Competing research groups want to emulate this model's predictions without having access to its internals.
(2) Application: Attackers can train a graph generator that is only based on surrogate model's predictions. As surrogate model gains more data and refines its predictions, it gradually aligns with the victim model's behavior. Over time, the surrogate model becomes a closer approximation of the victim model, and the attacker can extract valuable drug compound prediction knowledge.

Type III Attack: Exploiting Inconsistent Predictions:
(1) Real-World Scenario: Suppose there is a financial institution using a GNN for credit risk assessment. A competitor may be interested in understanding the decision-making process of this model to improve risk assessment strategy.
(2) Application: Attackers can simultaneously train two surrogate models, each of which attempts to mimic the victim model's behavior. If these surrogate models make inconsistent predictions for certain credit applications (e.g., one surrogate approves while the other rejects), it indicates that they have

Table 7: Architecture selection and accuracy of the victim model in two tasks.

		Dataset	Cora	Pubmed	A-Computer	OGB-Arxiv
Node Classification	GAT	Architecture	1-32	1-32	1-128	2-256
		Accuracy	79.56±0.76	77.06±0.54	80.09±1.67	54.66±0.36
	GCN	Architecture	3-128	2-256	2-256	3-256
		Accuracy	80.61±0.74	76.76±0.20	80.63±1.00	66.75±0.16
	SAGE	Architecture	2-256	3-128	2-64	3-256
		Accuracy	79.33±0.50	77.23±0.09	79.24±2.54	69.18±0.10
Link Prediction	GAT	Architecture	1-16	1-32	1-32	
		Accuracy	84.24±0.58	63.04±0.55	81.84±2.14	
	GCN	Architecture	1-128	2-256	3-128	
		Accuracy	89.70±0.30	88.10±0.18	93.84±0.35	
	SAGE	Architecture	2-32	2-32	5-32	
		Accuracy	92.47±0.50	86.91±0.29	86.42±2.96	

Table 8: Experimental Results on the Transductive Scenario for Node Classification Task. The table presents the mean and variance of 5 training runs.

$\mathcal{M}_V$	Dataset	Cora		Pubmed		A-Computers		OGB-Arxiv	
	$\mathcal{M}_S$(GCN)	Accuracy	Fidelity	Accuracy	Fidelity	Accuracy	Fidelity	Accuracy	Fidelity
GAT	Real Data	$83.68_{\pm 0.25}$	$94.47_{\pm 0.74}$	$78.62_{\pm 0.05}$	$92.21_{\pm 1.01}$	$83.72_{\pm 0.91}$	$91.72_{\pm 1.06}$	$56.73_{\pm 1.08}$	$89.23_{\pm 0.69}$
	Random Graph	$74.39_{\pm 2.95}$	$81.69_{\pm 1.87}$	$67.81_{\pm 1.59}$	$83.09_{\pm 2.29}$	$75.85_{\pm 3.84}$	$83.22_{\pm 2.92}$	$48.08_{\pm 3.74}$	$78.25_{\pm 3.89}$
	Attack I-$\tilde{\mathbf{E}}$	$83.41_{\pm 0.21}$	$94.12_{\pm 0.67}$	$78.37_{\pm 0.01}$	$91.78_{\pm 0.94}$	$83.48_{\pm 0.83}$	$91.03_{\pm 0.94}$	$55.89_{\pm 0.98}$	$88.73_{\pm 0.59}$
	Attack I-$\mathbf{A}_{cos}$	$83.26_{\pm 0.17}$	$93.63_{\pm 0.61}$	$78.46_{\pm 0.03}$	$90.93_{\pm 0.86}$	$83.35_{\pm 0.79}$	$91.23_{\pm 0.94}$	$56.02_{\pm 0.96}$	$88.35_{\pm 0.55}$
	Attack I-$\mathbf{A}_{fp}$	$83.37_{\pm 0.22}$	$93.79_{\pm 0.64}$	$78.51_{\pm 0.02}$	$91.11_{\pm 0.91}$	$82.92_{\pm 0.82}$	$91.35_{\pm 0.96}$	$56.48_{\pm 1.01}$	$88.60_{\pm 0.61}$
	Attack II-$\tilde{\mathbf{E}}$	$83.21_{\pm 0.19}$	$93.59_{\pm 0.63}$	$78.41_{\pm 0.01}$	$90.99_{\pm 0.89}$	$83.28_{\pm 0.81}$	$91.22_{\pm 0.98}$	$56.42_{\pm 0.99}$	$88.52_{\pm 0.57}$
	Attack II-$\mathbf{A}_{cos}$	$83.14_{\pm 0.15}$	$93.43_{\pm 0.59}$	$78.35_{\pm 0.02}$	$90.81_{\pm 0.83}$	$82.91_{\pm 0.77}$	$91.12_{\pm 0.92}$	$55.76_{\pm 0.53}$	$88.28_{\pm 0.51}$
	Attack II-$\mathbf{A}_{fp}$	$83.29_{\pm 0.18}$	$93.92_{\pm 0.62}$	$78.29_{\pm 0.03}$	$91.67_{\pm 0.88}$	$83.34_{\pm 0.78}$	$91.01_{\pm 0.95}$	$56.43_{\pm 0.96}$	$88.66_{\pm 0.54}$
	Attack III-$\tilde{\mathbf{E}}$	$83.55_{\pm 0.19}$	$93.87_{\pm 0.65}$	$78.38_{\pm 0.02}$	$91.82_{\pm 0.91}$	$83.39_{\pm 0.75}$	$91.08_{\pm 0.97}$	$56.55_{\pm 0.94}$	$88.81_{\pm 0.52}$
	Attack III-$\mathbf{A}_{cos}$	$83.45_{\pm 0.17}$	$94.12_{\pm 0.61}$	$78.36_{\pm 0.04}$	$91.95_{\pm 0.86}$	$82.68_{\pm 0.73}$	$91.16_{\pm 0.94}$	$56.49_{\pm 0.98}$	$88.71_{\pm 0.57}$
	Attack III-$\mathbf{A}_{fp}$	$83.62_{\pm 0.21}$	$94.01_{\pm 0.67}$	$78.42_{\pm 0.03}$	$91.71_{\pm 0.93}$	$83.56_{\pm 0.80}$	$91.24_{\pm 0.91}$	$56.57_{\pm 0.92}$	$88.96_{\pm 0.55}$
GCN	Real Data	$83.85_{\pm 0.28}$	$91.15_{\pm 0.85}$	$78.87_{\pm 0.17}$	$94.72_{\pm 0.51}$	$83.41_{\pm 0.68}$	$90.22_{\pm 0.76}$	$65.03_{\pm 0.96}$	$92.03_{\pm 0.82}$
	Random Graph	$72.19_{\pm 1.89}$	$80.53_{\pm 2.07}$	$63.56_{\pm 2.38}$	$81.39_{\pm 2.19}$	$72.08_{\pm 2.34}$	$77.92_{\pm 2.54}$	$50.28_{\pm 2.41}$	$81.37_{\pm 2.28}$
	Attack I-$\tilde{\mathbf{E}}$	$82.57_{\pm 0.48}$	$90.75_{\pm 1.19}$	$78.21_{\pm 0.05}$	$94.43_{\pm 0.62}$	$82.98_{\pm 0.62}$	$89.60_{\pm 0.99}$	$63.74_{\pm 0.73}$	$91.58_{\pm 0.37}$
	Attack I-$\mathbf{A}_{cos}$	$82.38_{\pm 0.61}$	$90.41_{\pm 1.06}$	$78.47_{\pm 0.14}$	$94.19_{\pm 0.52}$	$83.01_{\pm 0.57}$	$89.80_{\pm 0.83}$	$63.22_{\pm 0.84}$	$91.82_{\pm 0.54}$
	Attack I-$\mathbf{A}_{fp}$	$82.75_{\pm 0.67}$	$90.85_{\pm 0.96}$	$78.61_{\pm 0.24}$	$94.32_{\pm 0.46}$	$82.82_{\pm 0.75}$	$89.42_{\pm 0.72}$	$63.85_{\pm 0.91}$	$91.24_{\pm 0.67}$
	Attack II-$\tilde{\mathbf{E}}$	$82.84_{\pm 0.74}$	$91.42_{\pm 0.91}$	$78.69_{\pm 0.29}$	$94.61_{\pm 0.55}$	$82.89_{\pm 0.82}$	$89.97_{\pm 0.79}$	$64.11_{\pm 0.79}$	$91.92_{\pm 0.73}$
	Attack II-$\mathbf{A}_{cos}$	$82.91_{\pm 0.72}$	$91.23_{\pm 0.86}$	$78.68_{\pm 0.26}$	$94.56_{\pm 0.49}$	$82.98_{\pm 0.78}$	$89.81_{\pm 0.75}$	$63.84_{\pm 0.73}$	$91.66_{\pm 0.69}$
	Attack II-$\mathbf{A}_{fp}$	$82.98_{\pm 0.79}$	$91.71_{\pm 0.97}$	$78.81_{\pm 0.37}$	$94.45_{\pm 0.62}$	$83.07_{\pm 0.87}$	$90.18_{\pm 0.85}$	$64.38_{\pm 0.86}$	$92.02_{\pm 0.78}$
	Attack III-$\tilde{\mathbf{E}}$	$83.75_{\pm 0.86}$	$92.51_{\pm 1.10}$	$79.23_{\pm 0.45}$	$94.91_{\pm 0.79}$	$83.55_{\pm 0.96}$	$91.03_{\pm 0.96}$	$64.91_{\pm 0.97}$	$92.82_{\pm 0.97}$
	Attack III-$\mathbf{A}_{cos}$	$82.62_{\pm 0.33}$	$92.35_{\pm 1.06}$	$79.09_{\pm 0.55}$	$94.78_{\pm 0.72}$	$83.23_{\pm 0.92}$	$90.85_{\pm 0.92}$	$64.63_{\pm 0.95}$	$92.59_{\pm 0.92}$
	Attack III-$\mathbf{A}_{fp}$	$83.91_{\pm 0.97}$	$92.79_{\pm 1.15}$	$79.39_{\pm 0.64}$	$95.02_{\pm 0.86}$	$83.72_{\pm 1.01}$	$91.24_{\pm 1.01}$	$65.18_{\pm 1.02}$	$92.98_{\pm 1.01}$
SAGE	Real Data	$84.28_{\pm 0.16}$	$92.49_{\pm 0.39}$	$77.14_{\pm 0.81}$	$93.23_{\pm 0.95}$	$83.29_{\pm 0.28}$	$92.17_{\pm 0.49}$	$69.39_{\pm 0.53}$	$93.17_{\pm 0.79}$
	Random Graph	$71.99_{\pm 2.03}$	$81.13_{\pm 2.49}$	$68.27_{\pm 3.08}$	$85.28_{\pm 2.03}$	$75.38_{\pm 1.94}$	$83.72_{\pm 1.84}$	$52.34_{\pm 3.46}$	$80.97_{\pm 2.38}$
	Attack I-$\tilde{\mathbf{E}}$	$82.28_{\pm 0.33}$	$91.76_{\pm 0.64}$	$75.14_{\pm 0.93}$	$92.28_{\pm 0.89}$	$82.14_{\pm 0.48}$	$91.62_{\pm 0.68}$	$67.39_{\pm 0.71}$	$92.27_{\pm 0.88}$
	Attack I-$\mathbf{A}_{cos}$	$83.38_{\pm 0.25}$	$91.73_{\pm 0.55}$	$76.14_{\pm 0.78}$	$92.12_{\pm 0.79}$	$82.71_{\pm 0.39}$	$91.83_{\pm 0.59}$	$68.39_{\pm 0.64}$	$92.01_{\pm 0.79}$
	Attack I-$\mathbf{A}_{fp}$	$82.79_{\pm 0.41}$	$91.29_{\pm 0.49}$	$76.71_{\pm 0.68}$	$92.06_{\pm 0.86}$	$82.12_{\pm 0.33}$	$91.59_{\pm 0.45}$	$68.91_{\pm 0.53}$	$91.29_{\pm 0.78}$
	Attack II-$\tilde{\mathbf{E}}$	$83.14_{\pm 0.56}$	$91.84_{\pm 0.99}$	$77.01_{\pm 0.99}$	$92.81_{\pm 1.07}$	$82.63_{\pm 0.68}$	$92.17_{\pm 0.99}$	$69.49_{\pm 0.79}$	$91.89_{\pm 1.17}$
	Attack II-$\mathbf{A}_{cos}$	$83.43_{\pm 0.62}$	$91.57_{\pm 0.89}$	$77.35_{\pm 1.05}$	$92.32_{\pm 0.98}$	$82.74_{\pm 0.73}$	$92.45_{\pm 1.05}$	$70.22_{\pm 0.96}$	$92.11_{\pm 1.22}$
	Attack II-$\mathbf{A}_{fp}$	$83.68_{\pm 0.71}$	$91.92_{\pm 1.05}$	$77.87_{\pm 0.89}$	$92.69_{\pm 1.15}$	$82.95_{\pm 0.81}$	$92.79_{\pm 1.18}$	$70.63_{\pm 1.08}$	$92.83_{\pm 1.37}$
	Attack III-$\tilde{\mathbf{E}}$	$83.37_{\pm 0.56}$	$91.49_{\pm 0.82}$	$77.26_{\pm 0.92}$	$92.18_{\pm 0.88}$	$82.59_{\pm 0.66}$	$92.32_{\pm 0.86}$	$70.12_{\pm 0.78}$	$91.93_{\pm 1.02}$
	Attack III-$\mathbf{A}_{cos}$	$83.29_{\pm 0.53}$	$91.38_{\pm 0.77}$	$77.21_{\pm 0.88}$	$92.11_{\pm 0.83}$	$82.68_{\pm 0.69}$	$92.23_{\pm 0.81}$	$70.06_{\pm 0.75}$	$91.81_{\pm 0.99}$
	Attack III-$\mathbf{A}_{fp}$	$84.19_{\pm 0.37}$	$91.51_{\pm 0.83}$	$77.29_{\pm 0.94}$	$92.21_{\pm 0.90}$	$82.62_{\pm 0.67}$	$92.34_{\pm 0.88}$	$70.15_{\pm 0.80}$	$91.96_{\pm 1.05}$

learned different aspects of the victim model's decision criteria. By enforcing these inconsistencies and analyzing which applications trigger them, the attacker can gain insights into the victim model's complex risk assessment process.