Impact of Graph Structure on Membership-Inference Risk for Graph Neural Networks

Megha Khosla
Delft University of Technology
Delft, The Netherlands
m.khosla@tudelft.nl

Abstract

Graph neural networks (GNNs) are widely used for tasks such as node classification and link prediction, but their use in sensitive settings raises concerns about training-data leakage. Prior work on privacy leakage in GNNs largely borrows assumptions from non-graph domains, overlooking the role of graph structure. We argue for a graph-specific analysis of privacy risk and study how graph structure affects node-level membership inference. We formalize membership inference (MI) over node–neighborhood tuples and investigate two important dimensions: (i) training-graph construction and (ii) inference-time edge access. We compare *snowball sampling*, a structure-aware procedure, with *uniform random node sampling* for constructing training graphs. Our experiments show that snowball sampling often hurts generalization relative to random sampling due to its coverage bias. In contrast, allowing access to inter–train–test edges at inference improves test accuracy, reduces the train–test gap, while also having a strong and setting-dependent effect on membership advantage. These results show that graph structure directly shapes privacy risk. We further show that the generalization gap, measured as the performance difference between training and test nodes, is an *incomplete* proxy for membership inference risk: membership advantage can rise or fall independently of changes in this gap, with inference-time edge access often playing a crucial role. Theoretically, we show that for node-level tasks, standard privacy-auditing results based on membership inference do not directly carry over to inductive graph settings, because training and test nodes are structurally dependent rather than interchangeable. We release the code and data at https://github.com/PriXAI/GraphStructurePrivacyAnalysis-public.

Keywords

Membership-inference, Graph Neural Networks, Differential Privacy

1 Introduction

Graph Neural Networks (GNNs) are widely employed for learning on relational data, delivering state-of-the-art results in chemistry, recommender systems, knowledge graphs, and more [7, 27, 28]. However on the downside, GNNs have been shown to leak private information about the very graphs they are trained on [3–5, 9, 15, 21, 32–34]. Existing privacy risk studies for GNNs mainly adapt techniques from the i.i.d. scenarios [30], treating each example as independent and largely ignoring the graph structure that connects data points together.

We focus on membership inference (MI) risk: given access to a trained model, an adversary aims to determine whether a specific data instance was included in its training set. Studying membership inference is important not only because membership itself may constitute sensitive information, but also because it serves as a proxy for stronger privacy violations such as attribute inference or data reconstruction attacks. Intuitively, if an adversary cannot reliably determine whether a given instance participated in training, then reconstructing hidden properties or recovering parts of the training data becomes substantially harder.

In graph-based machine learning using graph neural networks, what is considered as a data instance depends on the task: node-level tasks view each node as an instance (e.g., predicting a protein's function in a PPI network), edge-level tasks treat each edge as an instance (e.g., recommending a friendship), and graph-level tasks consider the entire graph as a single instance (e.g., classifying a molecule's toxicity). Our study focuses on node-level tasks, where instances (being nodes) are inherently interdependent due to the existence of edges between them. This inter-instance dependency is largely absent from prior privacy analyses and leads us to ask:

> *How strongly does the amount of edge information available during training and at inference modulate node-level membership-inference risk?*

Training Graph Construction. To investigate this question, we start from the inductive setting of [21], where training and test node sets are disjoint and, during training, neither test nodes nor any train–test edges are available. A common way to instantiate such splits is to sample train/test nodes uniformly at random and use the induced subgraphs. While this preserves independence among sampled nodes, it is often unrealistic. For instance, random node sampling can produce isolated nodes when none of their neighbors are included, which undermines the rationale for graph-based learning.

Structure Based Sampling and Coverage Bias. A more plausible sampling strategy is *snowball sampling*, in which, starting from random seed nodes, we iteratively add a fixed number of randomly chosen neighbors until the desired training size is reached. Variants of snowball sampling are quite popular in real-world scenarios such as for building social-network graphs [19] and for contact tracing in healthcare [17]. Specifically following a snowball procedure help us in generating a well-connected training graph. Nevertheless, the generated sample is coverage-biased in the sense that the expansions tend to remain within a few regions, oversampling

This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license visit https://creativecommons.org/licenses/by/4.0/ or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.
Proceedings on Privacy Enhancing Technologies 2026(4), 628–650
© 2026 Copyright held by the owner/author(s).
https://doi.org/10.56553/popets-2026-0138

high-degree hubs while leaving peripheral or distinct communities under-represented. The resulting training distribution is therefore non-i.i.d. with respect to the full graph, motivating our first axis of inquiry: *How does biased coverage affect generalization to held-out nodes and, in turn, membership-inference risk?*

Edge Access at Inference Time. A second graph-specific axis is *edge access at inference*. In our node–neighborhood formulation, each query supplies a subgraph, and a GNN's prediction for a given node depends on that subgraph at query time. Even with a fixed train–test node split, adding or withholding edges can substantially alter the outputs of a frozen model. Unlike i.i.d. domains, an adversary may legitimately provide relational context when probing membership. We motivate this axis further through a practical example. Consider a disease-spread network constructed via contact tracing, where investigators start from an initial set of patients and expand the trace by following contacts. In such settings, even if an adversary has access to the broader population cohort, they may not know the realized tracing pattern, either partially or completely. We therefore ask: *How does the adversary's edge knowledge modulate MI success?*

In prior literature MI success is often linked to overfitting and generalization error [36]. Because the true data-generating distribution is unknown, practitioners use the empirical train–test *gap* as a surrogate. Yet prior work suggests that attacker performance does not always increase with this empirical gap in graph settings [21]. Here, the gap itself depends not only on which nodes are selected for train/test but also on which neighborhoods are presented at inference. This leads to our third guiding question: *What is the relationship between the generalization gap and membership-inference risk across training-graph constructions (random vs. snowball) and inference-time edge regimes?*

Exchangeability and Privacy Auditing. Finally, while we empirically studied membership advantage under different settings, we do not attempt to translate these results into a formal lower bound on the differential privacy budget of GNNs or to audit differential private GNNs, as is common in literature on privacy auditing [16]. The simple reason for that is such a formal connection cannot be always guaranteed under the differential privacy notion and when considering specific training graph construction schemes for inductive learning settings. To support our arguments, we adapt the definition of *statistical exchangeability* from [12] to GNNs by treating each example as a node–neighborhood tuple, and show that inductive train/test constructions, whether via random node sampling or snowball sampling, break exchangeability because neighborhoods are restricted to, and dependent on, the sampled training subgraph. This breaking of statistical exchangeability (as shown in Section 3.3) implies that the data-generating distribution may already leak information about node membership. In such cases, differential privacy, being oblivious to the data generating distribution, may not be auditable under practical node-level membership inference attacks.

2 Background

We provide background on the graph sampling strategies, Graph Neural Networks (GNNs), and membership advantage used throughout this work.

2.1 Graph Sampling Strategies

We employ two graph sampling strategies: *random sampling* and *snowball sampling*. These strategies determine not only which nodes are included in the training set but also the resulting graph structure, which in turn influences both model learning and the adversary's ability to infer membership. We chose random sampling as it is the simplest and the most widely used strategy in academic literature to construct train graphs for training graph neural networks. In contrast, as discussed in Section 1, structured sampling techniques such as snowball sampling are more commonly employed when constructing graphs from real-world data, especially in building social science surveys and in healthcare studies.

Random Sampling. Formally, given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of n nodes and $\mathcal{E}$ is the set of m edges, we randomly sample N nodes independently and uniformly to form the sampled node set $\mathcal{V}_S$. The training graph is then constructed by including an edge (u, v) if and only if both $u, v \in \mathcal{V}_S$ and $(u, v) \in \mathcal{E}$.

Snowball Sampling. Here we start from an initial set of nodes $\mathcal{V}^{(0)}$ (in our experiments we randomly chose 10 nodes from each class to form $\mathcal{V}^{(0)}$). At each stage i we choose k neighbors of each of the nodes in $\mathcal{V}^{(i-1)}$. This is equivalent to obtaining a sample of incident edges of $\mathcal{V}^{(i-1)}$ which we denote by $\mathcal{E}^{(i)}$. We construct $\mathcal{V}^i$ by adding only the new nodes discovered in the last stage. The process continues until the maximum number of nodes are sampled. The sampled train graph then is $\mathcal{G}_\mathcal{S} = (\mathcal{V}_S, \mathcal{E}_S)$ where $\mathcal{V}_S = \cup_i \mathcal{V}^{(i)}$ and $\mathcal{E}_S = \cup_i \mathcal{E}^{(i)}$

2.2 Graph Neural Networks

Graph Neural Networks (GNNs) are a class of machine learning models designed specifically to learn on graph-structured data. The typical input to a GNN consists of the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ and $\mathcal{E}$ denotes the set of nodes and edges (connecting pairs of nodes) respectively. In addition the node features matrix $\mathbf{X} = (\mathbf{x}_1^\top, \mathbf{x}_2^\top, \ldots, \mathbf{x}_n^\top)$ is provided where $\mathbf{x}_i$ is the input feature vector for node i. In the supervised setting, each node in the training set is associated with a one-hot encoded label vector $\mathbf{y}_i$, and the model is trained to predict these labels for unseen nodes.

At each GNN layer, a node's representation is updated by aggregating its own features along with the features of its neighbors. After this aggregation step, the node's representation is transformed using a non-linear function. More formally, let $\mathbf{x}_i^{(\ell)}$ denote the feature representation of node i at layer ℓ, and let $\mathcal{N}_i$ be the set of nodes in the 1-hop neighborhood of node i. The graph convolution operation at layer ℓ is defined as follows:

$$\mathbf{z}_i^{(\ell)} = \text{AGGREGATION}^{(\ell)}\left(\left\{\mathbf{x}_i^{(\ell-1)}, \left\{\mathbf{x}_j^{(\ell-1)} \mid j \in \mathcal{N}_i\right\}\right\}\right) \quad (1)$$

$$\mathbf{x}_i^{(\ell)} = \text{TRANSFORMATION}^{(\ell)}\left(\mathbf{z}_i^{(\ell)}\right) \quad (2)$$

At the final layer (denoted L), a softmax function is applied to the node representations to produce the predicted class probabilities for each node. This can be expressed as:

$$\hat{\mathbf{y}}_i \leftarrow \text{softmax}(\mathbf{z}_i^{(L)}\mathbf{\Theta}), \quad (3)$$

Here, $\hat{\mathbf{y}}_i \in \mathbb{R}^c$, where c is the number of classes, and $\mathbf{\Theta}$ is a learnable weight matrix. The jth element $\hat{\mathbf{y}}_i$ represents the predicted

probability that node i belongs to class j. We focus on three representative aggregation operations in GNNs, which serve as the basis for more complex aggregation methods. The concrete realizations of the aggregation and transformation operations for the three GNN architectures employed in this work are provided in Appendix A.

Train–Test Paradigms

Graph-based learning typically distinguishes between two train–test paradigms depending on whether test nodes are visible during training. In the **transductive** setting, the full graph $\mathcal{G}$ is available during training, although labels are only known for the train nodes. Consequently, test nodes can still influence learned representations through message passing and neighborhood aggregation. By contrast, in the **inductive** setting, train and test nodes are disjoint, and neither test nodes nor train–test edges are exposed during training. Following [21], we focus on the inductive setting, where the distinction between member and non-member nodes remains unambiguous.

2.3 Why Graph Structure Matters for GNN Learning and Privacy

We note that the aggregation mechanism of the GNNs (as depicted in (1)) makes the learned representation of a node directly dependent on the graph structure. Specifically, the representation of a node is influenced not only by its own features, but also by the features and connectivity of nodes in its neighborhood. Consequently, changes in the neighborhood structure, such as adding or removing edges and neighboring nodes, modify the information aggregated by the model and can therefore alter the resulting node representations and predictions.

2.3.1 Availability and Composition of Node Neighborhoods. The above discussed dependence on structure is relevant during both training and inference. During training, graph sampling strategies determine which neighborhoods are visible to the model, which nodes repeatedly participate in aggregation, and what structural patterns the model learns to rely on. As shown in Section 5.1, different sampling strategies can substantially alter the structural properties of the resulting training graph. For example, snowball sampling tends to bias the sample toward high-degree nodes because such nodes are more likely to be reached during recursive neighborhood expansion. In contrast, under uniform random node sampling, all nodes are sampled with equal probability, but the observed degree distribution in the induced subgraph shifts toward lower values because only edges between sampled nodes are retained.

2.3.2 Structural Properties and Representation Learning. The differences in availability and composition of node neighborhoods induced by different sampling strategies as discussed above directly affect GNN behavior. For instance, high-degree nodes typically receive richer neighborhood information during aggregation, whereas isolated or sparsely connected nodes provide limited context. Similarly, nodes connected to neighbors with similar labels or features are generally easier for GNNs to learn from. Consequently, the structural properties of the sampled training graph directly determine the node–neighborhood patterns observed during training and therefore influence model generalization. When the sampled graph does not adequately represent the structural diversity of the original graph, the learned representations may generalize unevenly across different regions of the network. In particular, structure-based sampling strategies such as snowball sampling can over-represent specific graph regions and connectivity patterns, leading to uneven graph coverage and increasing the behavioral gap between member and non-member nodes, which can potentially make membership inference easier.

2.3.3 Availability of Graph Structure at Inference Time. Since neighborhood aggregation is performed also during inference, the edges and neighborhoods accessible at query time directly influence the predictions of a frozen GNN. Consequently, even for a fixed trained model, predictions for the same node can vary depending on the graph structure provided during inference. In Section 5.2, we empirically analyze this effect by comparing the output distributions of the same trained model under different inference-time graph access regimes. We observe that changing only the neighborhood structure can significantly alter the output distributions produced by the model, which in turn can affect membership inference attack success.

2.4 Membership Advantage

To quantify privacy leakage under membership inference attacks, we follow [36] and use the notion of *membership advantage*. Intuitively, membership advantage measures how well an adversary can distinguish members from non-members beyond random guessing. An attacker with no useful information should behave similarly on both groups, resulting in an advantage close to zero, whereas a larger advantage indicates stronger membership leakage.

Formally, let $b \in \{0, 1\}$ denote the membership status of a queried instance, where $b = 0$ indicates that the instance belongs to the training set and $b = 1$ indicates a non-member instance. Let $\hat{b} \leftarrow \mathcal{A}(\cdot)$ denote the membership prediction of the adversary. The membership advantage of $\mathcal{A}$ is defined as

$$\mathrm{Adv}(\mathcal{A}) = \Pr(\hat{b} = 0 \mid b = 0) - \Pr(\hat{b} = 0 \mid b = 1).$$

that is, the difference between the true positive rate and false positive rate of the attacker.

In practice, the attacker outputs a membership score $s(x) \in [0, 1]$, representing the predicted likelihood that an input belongs to the training set. Given a threshold τ, the attacker predicts membership whenever $s(x) \geq \tau$. The empirical membership advantage at threshold τ is therefore defined as:

$$\widehat{\mathrm{Adv}}_\tau(\mathcal{A}) = \widehat{\mathrm{TPR}}(\tau) - \widehat{\mathrm{FPR}}(\tau),$$

where $\widehat{\mathrm{TPR}}(\tau)$ and $\widehat{\mathrm{FPR}}(\tau)$ denote the empirical true positive rate and false positive rate of the attacker at threshold τ, respectively.

We then compute the empirical membership advantage as the maximum difference between the true positive rate and false positive rate across all thresholds:

$$\widehat{\mathrm{Adv}}(\mathcal{A}) = \max_\tau \left(\widehat{\mathrm{TPR}}(\tau) - \widehat{\mathrm{FPR}}(\tau) \right).$$

Equivalently, this corresponds to the maximum vertical separation between the ROC curve of the attacker and the diagonal random-guessing baseline.

3 Node-level Membership Inference

We study the problem of node-level membership inference in Graph Neural Networks (GNNs), where an adversary aims to determine whether a queried node was part of the training graph used to train a target model. To formalize node-level membership inference in GNNs, we first explicitly define the input instance corresponding to a node v. Let $\mathbf{x}_v \in \mathbb{R}^d$ denote the feature vector of node v, $y_v \in \mathcal{Y}$ its label, and $\mathcal{N}^L_{\text{strategy}}(v)$ its L-hop neighborhood computed according to a neighborhood selection strategy (e.g., induced neighborhood, sampled neighbors, or full graph). Given a training graph $\mathcal{G}_S = (\mathcal{V}_S, \mathcal{E}_S) \sim \mathcal{S}(\mathcal{G}, n)$, constructed by sampling n nodes from the original graph $\mathcal{G}$ according to a sampling strategy $\mathcal{S}$, the node-level input representation presented to the model is defined as

$$z_v^{\mathcal{G}_S} := \left(\mathbf{x}_v, y_v, \mathcal{N}^L_{\text{strategy}}(v), \left\{\mathbf{x}_u : u \in \mathcal{N}^L_{\text{strategy}}(v)\right\}\right). \tag{4}$$

3.1 Node-Level Membership Inference Experiment

We now define the node-level membership inference (MI) experiment while making explicit the role of graph sampling and neighborhood construction. For notational convenience, we restrict ourselves to multi-class settings where each node v is associated with a single ground truth label y_v.

Experiment 1. *(Node-Level Membership Inference* $\text{Exp}(\mathcal{A}, \Phi, n, \mathcal{S})$*) Let $\mathcal{A}$ be an adversary, Φ a learning algorithm, $n \in \mathbb{N}$ a sample size, and $\mathcal{S}$ a graph sampling strategy.*

1. *Sample a subgraph $\mathcal{G}_S = (\mathcal{V}_S, \mathcal{E}_S) \sim \mathcal{S}(\mathcal{G}, n)$.*
2. *Train a GNN model $\Phi_S = \Phi(\mathbf{X}_S, \mathcal{G}_S)$.*
3. *Choose a bit $b \leftarrow \{0, 1\}$ uniformly at random.*
4. *If $b = 0$, sample a member node $v \sim \mathcal{V}_S$; otherwise sample a non-member node $v \sim \mathcal{V} \setminus \mathcal{V}_S$.*
5. *Construct the node tuple*

$$z_v := \left(\mathbf{x}_v, y_v, \mathcal{N}^L(v), \left\{\mathbf{x}_u : u \in \mathcal{N}^L(v)\right\}\right),$$

   where $\mathcal{N}^L(v)$ denotes the L-hop neighborhood of node v computed based on the adversary's graph knowledge $\mathcal{G}_{Adv}$.
6. *The adversary outputs a membership prediction based on*

$$\mathcal{A}(\mathbf{x}_v, z_v, \Phi_S, \mathcal{G}_{Adv}).$$

Given the trained model Φ_S, target node v, and its corresponding node tuple z_v constructed using adversarial graph knowledge from some graph $\mathcal{G}_{Adv}$, the goal of the adversary is to determine whether v was part of the sampled training graph $\mathcal{G}_S$.

3.2 Differences from Classical MI Settings

Compared to classical membership inference attacks [36], the graph setting introduces two important differences. First, unlike classical membership inference settings where inputs are independent feature vectors, the learning algorithm here operates on the node–neighborhood representations defined in (4). Consequently, the sampling process used to construct the training graph itself becomes part of the MI experiment, as reflected in Step 1 and 2 of Experiment 1.

Second, the adversary's capabilities depend on access to graph structure. Since GNN predictions depend on neighborhood aggregation, the queried input tuple constructed in Step 5 of Experiment 1 explicitly includes the neighborhood $\mathcal{N}^L(v)$, constructed using adversary's knowledge of graph $\mathcal{G}_{Adv}$. Consequently, the validity and strength of the threat model directly depend on assumptions about the adversary's knowledge of the underlying graph structure as also illustrated in the following example.

Example. A social scientist or healthcare provider may sample part of a population network to study voting behavior or disease spread and subsequently train a GNN on the collected subgraph. In such settings, the broader population network may itself be publicly or partially observable, while the specific subset of individuals selected into the study remains private. An adversary may therefore attempt to infer which individuals were included in the training graph. At one extreme, the adversary may have access to the full underlying graph structure and all node relationships. At the other extreme, the adversary may only observe node-level features without any knowledge of graph connectivity or neighborhood structure. In practice, however, realistic adversarial knowledge often lies somewhere in between these two extremes, where the adversary possesses only partial or noisy information about the neighborhood connectivity.

3.3 Statistical Implications of Neighborhood-Based Input

An important distinction in graph settings is that, due to the explicit use of neighborhood information in GNNs (as reflected in Equation (4)), train and test instances are not always statistically exchangeable. Existing results that relate empirically measured membership advantage to lower bounds on a model's differential privacy budget [36] rely on this exchangeability assumption. As discussed by [12], such guarantees require train and test instances to be drawn from the same underlying distribution and remain interchangeable under permutation without altering the joint distribution. In graph domains, however, node representations depend on shared edges and overlapping neighborhoods, introducing structural dependencies between instances and thereby violating statistical exchangeability.

To support our claim we begin by reproducing and adapting the definition of statistical exchangeability from [12] to the setting of message passing graph neural networks (GNNs) as studied in this work.

Definition 3.1 (Statistical Exchangeability in MI for GNNs). Let $\mathcal{D}$ denote a joint distribution over n member samples $\{z_1, \ldots, z_n\}$ and a non-member sample z_{n+1}, where each sample z_v corresponds to a tuple:

$$z_v := (\mathbf{x}_v, y_v, \mathcal{N}^L_{\mathcal{G}'}(v), \{\mathbf{x}_u : u \in \mathcal{N}^L_{\mathcal{G}'}(v)\}),$$

with $\mathbf{x}_v$ and y_v being the feature vector of node v and its label respectively, and $\mathcal{N}^L_{\mathcal{G}'}(v)$ denoting the L-hop neighborhood of v in a graph view $\mathcal{G}'$. The graph $\mathcal{G}'$ may correspond to the full graph $\mathcal{G}$, or to a subgraph induced by the sampled training node set with

edges included according to a specific sampling scheme such as random or snowball sampling.

We say that $\mathcal{D}$ is *statistically exchangeable* if the joint distribution over all samples $\{z_1, \ldots, z_{n+1}\} \sim \mathcal{D}$ is invariant under permutations of sample indices. That is, for any permutation $\sigma : [n+1] \to [n+1]$, it holds that:

$$\Pr(z_1, \ldots, z_{n+1}) = \Pr(z_{\sigma(1)}, \ldots, z_{\sigma(n+1)}). \tag{5}$$

In the following, we show that statistical exchangeability cannot, in general, be guaranteed in graph learning settings with GNNs. In particular, this assumption is violated in inductive settings under both random and snowball sampling.

Theorem 3.2. *Let $\mathcal{D}$ be a joint distribution over n member samples $\{z_1, \ldots, z_n\}$ and a non-member sample z_{n+1}, where each sample z_v is defined as in Definition 3.1, and where neighborhoods $\mathcal{N}^L_{\mathcal{G}'}(v)$ are computed over a graph view $\mathcal{G}'$ constructed via a node sampling scheme. Then $\mathcal{D}$ is not always guaranteed to be statistically exchangeable if $\mathcal{G}'$ is a subgraph constructed over the a sampled set of training nodes $S \subset V$, and $\mathcal{N}^L_{\mathcal{G}'}(v)$ is restricted to lie within S.*

Proof. We consider two inductive sampling schemes as studied in this work: random node sampling and snowball sampling.

Case 1: Random Node Sampling. Let $\mathcal{G}'$ denote the subgraph induced over the training node set $S = \{x_1, \ldots, x_n\}$, and let the joint sample sequence be $Z = (z_1, \ldots, z_n, z_{n+1})$, where each z_i is defined as in Definition 3.1.

Now consider a permutation $Z^\sigma = (z_{\sigma(1)}, \ldots, z_{\sigma(n+1)})$ in which the non-member node x_{n+1} replaces some member node $x_i \in S$. The resulting training set becomes:

$$S' = \{x_1, \ldots, x_{i-1}, x_{n+1}, x_{i+1}, \ldots, x_n\},$$

and we define the new induced subgraph as $\mathcal{G}''$. Since neighborhoods $\mathcal{N}^L_{\mathcal{G}}(v)$ depend on the underlying graph, we have:

$$\mathcal{N}^L_{\mathcal{G}'}(v) \neq \mathcal{N}^L_{\mathcal{G}''}(v) \quad \text{for some } v \in S \cup \{x_{n+1}\},$$

unless both the exchanged nodes are isolated in the original graph $\mathcal{G}$. Thus, the permuted tuple Z^σ becomes incompatible with the graph structure $\mathcal{G}'$, and so:

$$\Pr(Z^\sigma \mid \mathcal{G}') = 0, \quad \text{while} \quad \Pr(Z \mid \mathcal{G}') > 0.$$

Since the graph view $\mathcal{G}'$ is generated by the sampling process with non-zero probability, i.e., $\Pr(\mathcal{G}') > 0$, it follows that the joint distribution is not invariant under permutation:

$$\Pr(Z) \neq \Pr(Z^\sigma),$$

violating statistical exchangeability.

Case 2: Snowball Sampling. Let $\mathcal{G}'$ be the subgraph generated using a snowball sampling process starting from a seed node set while sampling a fixed number of neighbors for each selected node in each iteration. In this setting, the set of sampled nodes and edges depends on the order in which nodes are traversed.

Now consider a permutation Z^σ where the non-member node x_{n+1} replaces a member node x_i. The resulting training set becomes:

$$S' = \{x_1, \ldots, x_{i-1}, x_{n+1}, x_{i+1}, \ldots, x_n\}.$$

Snowball sampling proceeds in layers by expanding neighbors of already sampled nodes. This means that whether a node is even included in the final subgraph depends on the specific sequence in which other nodes were included before it. Therefore, replacing x_i with x_{n+1} can not only alter neighborhoods but may also cause some nodes that were previously sampled to no longer appear in the new corresponding graph $\mathcal{G}''$, and vice versa.

As a result, the neighborhood $\mathcal{N}^L_{\mathcal{G}''}(v)$ and even the support of Z^σ may differ fundamentally (in terms of node features) from that of Z, and under a fixed $\mathcal{G}'$, we have:

$$\Pr(Z^\sigma \mid \mathcal{G}') = 0, \quad \text{while} \quad \Pr(Z \mid \mathcal{G}') > 0.$$

Since the graph view $\mathcal{G}'$ is generated by the sampling process with non-zero probability, i.e., $\Pr(\mathcal{G}') > 0$, it follows that the joint distribution is not invariant under permutation:

$$\Pr(Z) \neq \Pr(Z^\sigma),$$

violating statistical exchangeability. Thus, snowball sampling also violates statistical exchangeability due to its sequential and history-dependent nature. □

From the above discussion, we conclude that in inductive train-test splits, where the neighborhood structure of the training graph is determined by the set of sampled training nodes, statistical exchangeability between training and test samples cannot be always guaranteed. In contrast, in transductive settings, where the full graph $\mathcal{G}$ is accessible and fixed during training thus decoupling neighborhood structure from the sampled training nodes, statistical exchangeability holds when the training nodes are sampled randomly.

However, if snowball sampling is used to select training nodes, even when the full graph $\mathcal{G}$ is utilized during training, statistical exchangeability is still violated. This is because node inclusion under snowball sampling depends on the initial seed set and the traversal order, making the set of training nodes itself order-dependent. In other words, replacing even a single node in the sampled training set can make the resulting set structurally implausible under the sampling process.

4 Attack Model and Datasets

We now describe the attack model and the datasets employed in our experimental evaluation.

4.1 Attack Model

Existing works [5, 10, 21] typically employ a shadow model based [30] strategy to build the attack model. In particular, a shadow model is trained to mimic the target model. The shadow model employs a shadow dataset usually assumed to be sampled from the same distribution as the training dataset of the target model. The attack model is designed as a binary classification model, which maps the output prediction probabilities of the shadow model on the shadow dataset $\mathcal{D}_{shadow}$ to the membership of the corresponding input member and non-member nodes.

To isolate the impact of the shadow dataset quality and different shadow model training methods from the influence of graph structure on the performance of membership inference (MI), we entirely skip the step of building the shadow model. Instead, we directly train the attack model using the membership status of r fraction of both the true member and non-member instances to the attackers,

where $r \in \{0.1, 0.2, 0.5, 0.8\}$, and evaluate membership prediction on the remaining nodes.

We employ a 2-layer multilayer perceptron with ReLU activation as our attack model. The input to the attack model is the output prediction vector of the target model corresponding to the query node concatenated with the cross entropy loss value computed over the query node. The attack model outputs the probability that the queried node belongs to the training graph. Formally, let $\mathbf{p}(v)$ represent the output prediction vector for the query node v, and let $\mathcal{L}(v)$ denote the cross-entropy loss computed over v. The input to the attack model corresponding to node v is the concatenation of these two values, which can be written as:

$$\mathbf{x}_{\text{attack}} = [\mathbf{p}(v) \mid \mathcal{L}(v)].$$

Remark 1. Our goal is *not* to propose a new attack but to quantify how *edge structure* (employed during training-graph construction and inference-time edge access) affects membership advantage. The absolute success rate of any attack may change with attacker strength, model choices or defences (e.g., DP, calibration). Our focus, however, is on the *modulation* induced by edge structure. For this purpose, our setup using true membership for a large amount of data suffices to reveal the direction and relative magnitude of these edge-driven effects.

Remark 2. With slight abuse of terminology and for brevity we interchangeably refer to the prediction vector output by GNN for a node query as the **node's posterior**.

4.2 Datasets

We perform our investigations on three popular graph datasets, namely Cora [29], Chameleon [23] and PubMed [29] datasets. Cora and PubMed are citation datasets where each research article is a node, and there exists an edge between two articles if one article cites the other. Each node has a label that shows the article category. In Cora, the features of each node are represented by a 0/1-valued word vector, which indicates the word's presence or absence from the article's abstract. In PubMed, the node features are represented by the TF/IDF weighted word vector of the unique words in the dictionary.

Chameleon is a Wikipedia network dataset. Nodes represent articles from the English Wikipedia, edges reflect mutual links between them. Node features indicate the presence of particular nouns in the articles. The nodes were classified into 5 classes in terms of their average monthly traffic. All datasets are employed for the task of multi-class node classification.

The statistics of the datasets are provided in Table 1. Here, we compute average label homophily which measures how often adjacent nodes share the same label. Specifically, for each node v with degree at least 1 we compute its label homophily as the fraction of its 1-hop neighbors which has the same label as v. We then compute average label homophily of the dataset (presented under Homophily in Table 1) as the average over label homophily of all non-isolated nodes. High homophily corresponds to connected nodes tending to share the same labels, as observed in datasets such as Cora and PubMed, whereas in heterophilic datasets such as Chameleon, connected nodes tend to have dissimilar labels.

Table 1: Dataset statistics. $|\mathcal{V}|$ and $|\mathcal{E}|$ denote the number of nodes and (undirected) edges respectively, $|C|$ is the number of classes, d is the dimension of the feature vector. Under Homophily we present the average of label homophily of all non-isolated nodes in the dataset.

	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert C\rvert$	d	Homophily
Cora	2,708	5278	7	1,433	0.8252
Chameleon	2,277	31,371	5	2,325	0.2471
PubMed	19,717	44,324	3	500	0.7924

4.3 Settings for Generating Training Splits.

We sampled 5 train graphs each using snowball sampling and random sampling. For all datasets for all splits we fix the number of nodes in the train set to be 10% and 50% of the total nodes. In addition to the usual 50% split size adopted in evaluations in existing works we chose to include the 10% split size to simulate realistic settings where the number of member data points (training nodes) is typically much smaller than the overall population. For snowball sampling we build the initial node set $\mathcal{V}^{(0)}$ by selecting 10 nodes from each class randomly. We fix $k = 3$ (number of maximum neighbors to be sampled from each selected node) for all datasets.

In Table 2 we provide mean of node degree over the train splits and the corresponding standard deviation as well as the average degree of the original graph.

Table 2: Average degree of *train graphs* when 10% or 50% of nodes are sampled using snowball (k=3) or random and the underlying full graph without any sampling. Values are mean ± std over 5 splits.

	Sampling Type	Cora	Chameleon	PubMed
10%	Snowball	1.68 ± 0.046	1.93 ± 0.052	2.55 ± 0.058
	Random	0.32 ± 0.063	2.79 ± 0.495	0.41 ± 0.034
50%	Snowball	3.18 ± 0.051	4.91 ± 0.143	3.55 ± 0.013
	Random	2.00 ± 0.111	13.52 ± 0.741	2.21 ± 0.029
	Full Graph	3.90 ± 0.00	27.55 ± 0.00	4.496 ± 0.00

5 Influence of Sampling Strategy and Edge Access

Before quantifying their effect on membership advantage (Sec. 6), we *motivate and justify our design choice* to vary two graph–specific factors *training-graph sampling* and *inference-time edge access* by systematically characterizing their impact on (i) the properties of sampled nodes, (ii) the distribution of model outputs, and (iii) distribution of true class confidence scores.

5.1 Characteristics of Sampled Nodes

From Table 2 we observe that snowball sampling leads to higher connectivity, except in the case of chameleon, where random samples have higher edge density. The reason here is that under snowball sampling, we impose (in our experiments) a cap on the maximum

number of sampled neighbors, which truncates the neighborhoods of high-degree nodes. This limits the resulting edge density. In contrast, random node sampling does not impose such a restriction, so the average degree of the sampled subgraph can more directly reflect the high average degree of the original graph. In general, as we argue below snowball sampling biases the sample toward high-degree nodes, which affects both how representative the sample is and how GNN models behave.

With uniform random sampling, as studied in the current work, each node in the original graph is included in the sample independently with equal probability p. This implies that high-degree and low-degree nodes are sampled with equal likelihood. However, in the induced subgraph formed by the sampled nodes, high-degree nodes tend to appear with lower observed degrees. Specifically, a node of degree ℓ in the original graph will retain only those edges that connect to other sampled nodes. The probability that exactly k of its ℓ neighbors are also sampled follows a binomial distribution with parameters ℓ and p. Let P_ℓ denote the probability that a node has degree ℓ in the original graph, and let q_k denote the probability that a node has degree k in the sampled graph. Then, the degree distribution in the sampled graph is given by:

$$q_k = \sum_{\ell=k}^{\infty} \binom{\ell}{k} p^k (1-p)^{\ell-k} P_\ell, \tag{6}$$

This expression accounts for all nodes of degree $\ell \geq k$ in the original graph and computes the probability that exactly k of their neighbors are also sampled. As p decreases, the probability of retaining a high fraction of neighbors drops, leading to a shift in the degree distribution toward lower values.

In contrast, snowball sampling where sampling proceeds by recursively including neighbors of already sampled nodes tends to favor high-degree nodes. This is because such nodes have more opportunities to be reached during the expansion process, and once included, are more likely to bring in additional neighbors. To mitigate the rapid expansion caused by high-degree nodes, which, in snowball sampling, can otherwise concentrate the sampled subgraph in very small regions of the graph—we cap the number of neighbors sampled from each node. This constraint also helps control the average degree of the resulting subgraph, keeping it close to a predefined threshold. Nevertheless, the **probability of selecting a node in the training set increases with its degree** under snowball sampling, whereas it remains uniform under random sampling. An adversary who knows the sampling mechanism can exploit this asymmetry by assigning higher prior membership probabilities to high-degree nodes, even before observing the model output. Moreover, snowball-sampled training graphs tend to concentrate on specific regions of the graph determined by the reachability of the initial seed nodes. This restricted coverage of the graph, as also observed in our experiments, increases the gap between the model's behavior on member and non-member nodes.

5.2 Node Neighborhoods and Output Distributions

In addition to the influence of the edge structure during training, the output distributions of a GNN for the same set of query nodes can vary at inference time, even when the model parameters are fixed. This variation arises from the inherent *message-passing* or *aggregation* operation in GNNs (cf. Eq. (1)), which aggregates features from the node's L-hop neighborhood. We consider three types of graph access at inference time:

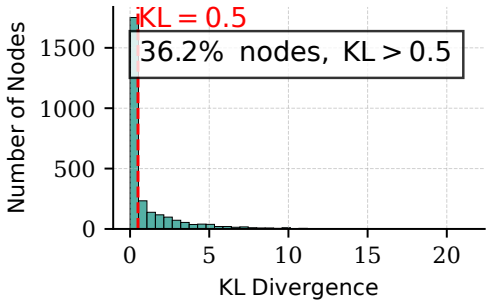

Figure 1: Distribution of KL divergence among posterior distribution of nodes on Cora dataset comparing the cases when all edges and none of the edges were used during inference. The model (Gcn) was trained on a snowball sampled split with 50% of the nodes in the train set.

- Original: Only the original train edges and test edges are used; train and test sets are edge-disjoint.
- FullGraph: All edges from the underlying graph—including edges between train and test nodes—are used during inference.
- NoEdges: The graph structure is not used at all during inference.

Let Φ be a fixed trained GNN model, and let v be a query node. Let $\mathcal{N}^L_{\mathcal{G}_1}(v)$ and $\mathcal{N}^L_{\mathcal{G}_2}(v)$ denote the L-hop neighborhoods of node v in two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ that differ in edge structure (e.g., different sampling strategies or inference-time neighborhood truncation). Then, the model's output for v under these two neighborhoods is given by:

$$\mathbf{y}_v^{(1)} := \Phi(v; \mathcal{N}^L_{\mathcal{G}_1}(v)) \quad \text{and} \quad \mathbf{y}_v^{(2)} := \Phi(v; \mathcal{N}^L_{\mathcal{G}_2}(v)).$$

We compute the KL-divergence between the two prediction distributions to capture the discrepancy in the model's output for the same node under different neighborhood contexts:

$$\mathcal{D}(v; \mathcal{G}_1, \mathcal{G}_2) := D_{KL}(\mathbf{y}_v^{(1)} || \mathbf{y}_v^{(2)}).$$

This divergence can be non-zero even for nodes in the training set, highlighting that inference-time neighborhood construction can directly and significantly affect the resulting node representations and, consequently, the model's predictions. An extreme illustration of this occurs when all neighbors are included during inference versus when all edges are ignored resulting in entirely different aggregation contexts and hence different predictions. In Figures 1 and 2, we show how the output posterior distribution, that is, the predicted probability vector, changes for the same nodes and the same model when only the neighborhood structure is varied. The large differences in model posteriors for the same nodes imply that attack success rates can vary substantially for those nodes depending on the adversary's knowledge of the edge structure.

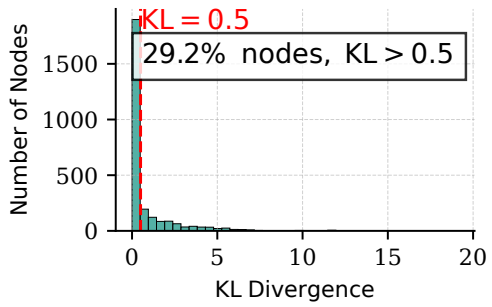

Figure 2: Distribution of KL divergence among posterior distribution of nodes on Cora dataset comparing the cases when all edges and none of the edges were used during inference. The model (Gcn) was trained on a randomly sampled split with 50% of the nodes in the train set.

5.3 Distribution of True Class Confidence

For each node, we map the prediction confidence $p \in (0, 1)$ to the *log-odds* (logit)

$$z = \text{logit}(p) = \log\left(\frac{p}{1-p}\right).$$

This places probabilities on an unbounded, symmetric scale ($z = 0$ at $p = 0.5$; $z > 0$ iff $p > 0.5$), which spreads out extreme values near 0/1 and facilitates distributional comparisons [2]. To avoid infinities at $p \in \{0, 1\}$, we clamp $p \leftarrow \min(\max(p, \varepsilon), 1 - \varepsilon)$ with $\varepsilon = 10^{-6}$ before applying the transform. (The inverse mapping is the sigmoid $p = \sigma(z) = 1/(1 + e^{-z})$.)

In Figure 3 we plot the distribution for logit transformed true class confidence scores for Cora with GraphSage under different edge access settings. Note that the trained model as well as the membership and non-membership status of nodes is fixed according to the original split. Only edge access is changed during inference to obtain prediction scores. We observe that the scores provide greater distinguishability when edges from the original split are included at inference compared to the other settings.

6 Graph Structure and Membership Advantage

After presenting qualitative discussion and preliminary evidence in favour of our argument to study the role of structure we now present large-scale quantitative results on how edge structure impacts membership advantage in GNNs. We begin by analyzing the effect of *training-graph construction* and *inference-time edge access*, on predictive performance (Section 6.2), and then study how the resulting train–test performance gap correlates with membership advantage (Section 6.3).

6.1 Experimental Setup

For each dataset we perform our experiments on 5 training splits constructed using snowball sampling and random sampling as described in Section 4. The attack model is a 2-layer MLP trained on an r fraction of member and non-member data points with their true labels. By varying $r \in \{0.1, 0.2, 0.5, 0.8\}$, we construct attack models with different levels of adversarial knowledge, allowing us to study whether the structural effects observed in our investigation remain consistent under different attacker strengths. For each of the training-test data split of the target model, we generate 3 random splits for each of the attack models. The attack results corresponding to each r are then summarized over a total of 15 runs for each dataset. We employ Gcn [18], Gat [31] and GraphSage [8] as the GNN models. All GNN models are evaluated with exactly the same data splits. Hyperparameter details of our implementation are provided in Appendix B.

We compute the performance gap as the percentage decrease in test accuracy relative to the training accuracy, i.e.,

$$\Delta_{gen} = \frac{ACC_{train} - ACC_{test}}{ACC_{train}} \times 100.$$

We employ performance gap between the train and test sets to approximate generalizaton gap. We consider three types of graph access at inference time:

- Original: Only the original train edges and test edges are used; train and test sets are edge-disjoint.
- FullGraph: All edges from the underlying graph—including edges between train and test nodes—are used during inference.
- NoEdges: All edge connections, that is, the graph structure is completely discarded during inference.

6.2 How do Graph Sampling and Edge Access Affect Performance Gap?

In figures 4, 5, 6 we compare the performance gap of the three GNNs under two train-graph construction strategies and the different edge access levels for Cora, Chameleon and PubMed datasets. Detailed train and test accuracy scores for all datasets are presented in Tables 3–4 (Cora), 5–6 (Chameleon), and 7–8 (PubMed) in Appendix C.

6.2.1 Effect of Sampling Type. When comparing different sampling strategies for constructing the training graph, we would expect higher performance gap when train graphs are snowball sampled. The rational behind this is that with random sampling, despite disrupting the graph structure, we obtain a i.i.d. node distribution. In contrast, snowball sampling introduces bias by expanding from a seed set, increasing the likelihood of sampling nodes that are structurally similar or closely connected. While the expected trend is observed over all datasets there are a few exceptions when 10% of nodes were used for training. Overall the observed performance gaps are highest for low homophilic dataset Chameleon and lowest for PubMed.

6.2.2 Effect of Edge Access. The performance gap varies depending on the level of graph access. The gap is usually smallest when all edges are used highlighting the benefit of including inter-train-test edges in improving test-time predictions. For Chameleon, although the performance gap is lower in the FullGraph setting, both train and test accuracies decrease for Gcn and Gat (with GraphSage as the main exception). This is expected in a low-homophily graph, where neighboring nodes are less likely to share labels: adding more edges can therefore hurt the message-passing behavior of Gcn and Gat, which implicitly smooth representations across adjacent nodes. The case of GraphSage is particularly interesting, as the accuracy on training nodes changes little across edge-access

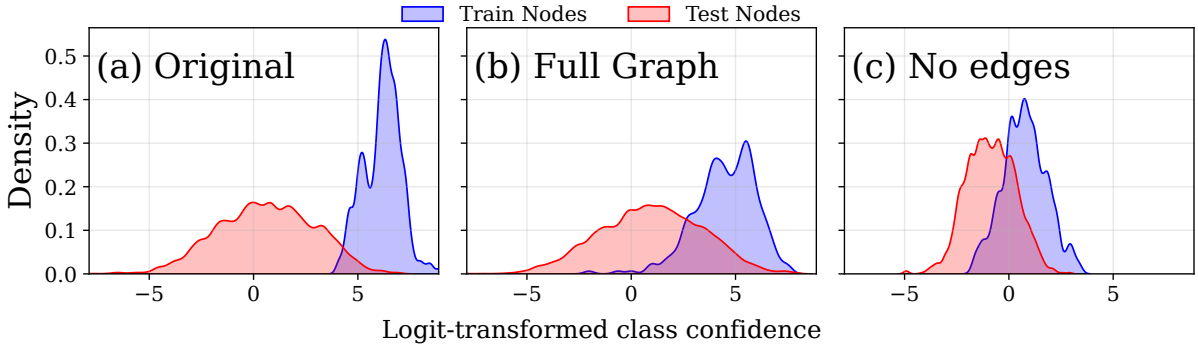

Figure 3: Effect of edge structure on train–test separability of class confidence (log-odds of the true-class prediction probability) for a GraphSage model. The model was trained over Cora dataset with 10% of nodes sampled for training using snowball sampling. We compare three inference graphs: (a) *Original* (train/test edge sets disjoint), (b) *Full graph* (all edges available), and (c) *No edges*. Train (blue) and test (red) distributions are most separated in (a), indicating stronger distinguishability; in (b) and (c) they overlap more, indicating weaker distinguishability.

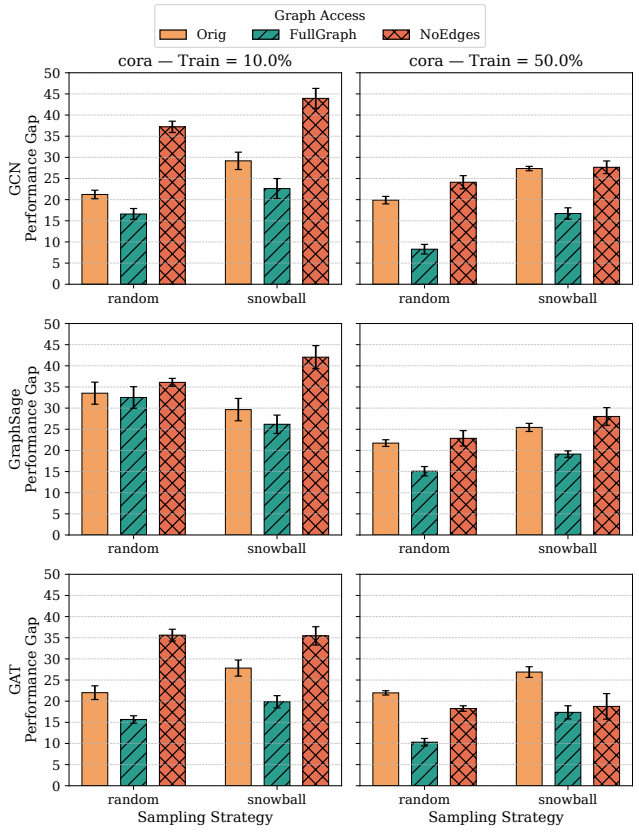

Figure 4: Performance gap (in %) under different edge access settings for Cora. The left plot corresponds to a training size of 10% of nodes, while the right plot corresponds to a training size of 50% of nodes.

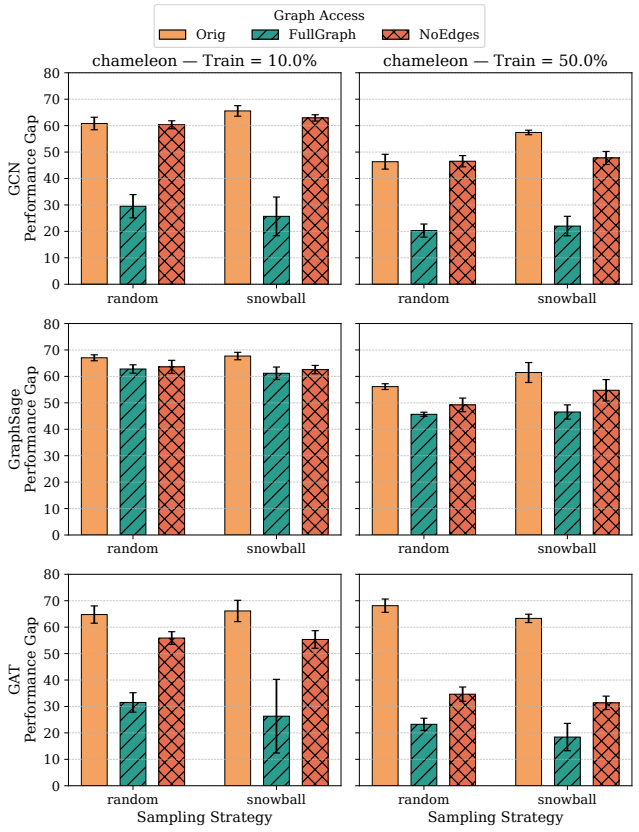

Figure 5: Performance gap (in %) under different edge access settings for Chameleon. Left plot corresponds to a training size of 10% of nodes, while the right plot corresponds to a training size of 50% of nodes.

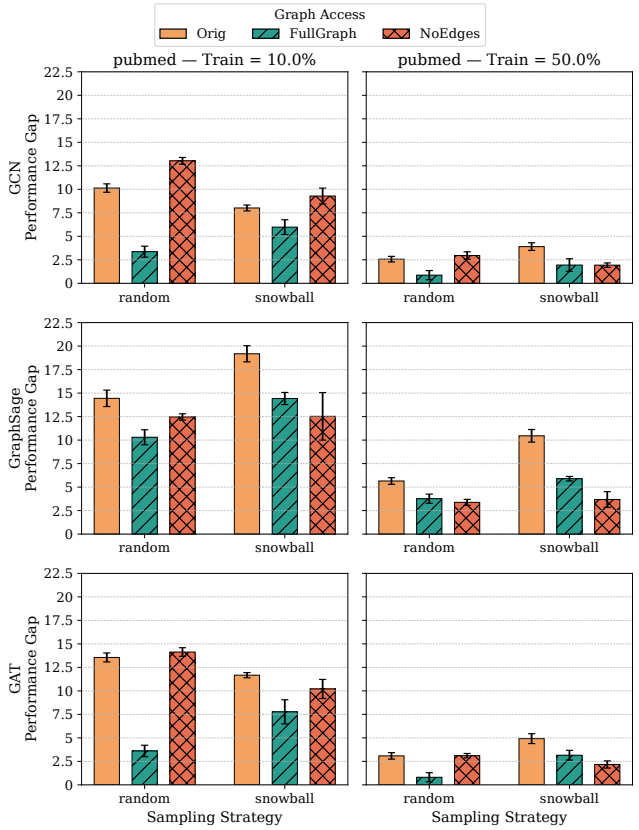

Figure 6: Performance gap (in %) under different edge access settings for PubMed. The left plot corresponds to a training size of 10% of nodes, while the right plot corresponds to a training size of 50% of nodes.

settings. We hypothesize that this is due to its aggregation mechanism, which concatenates the ego-node representation with the aggregated neighbor representation rather than averaging them together. As we show later, this distinction also has important consequences for privacy leakage when we examine membership advantage across these settings. We also see some exceptions in PubMed where in FullGraph setting the performance gap is sometimes higher than that of NoEdges setting. At the moment we do not have sound explanation for the phenomena. What is important, however, is that different assumptions about edge access at inference time do induce different levels of performance gap, which in turn can lead to different levels of privacy leakage.

6.2.3 Effect of Sampling Size . When the number of training nodes is increased from 10% to 50% a decrease in performance gap as expected is observed (with minor exceptions) for all models at different graph access levels and train sampling strategies.

6.3 How does Performance Gap Translate to Membership Advantage?

Figures 7 and 8 show scatter plots of performance gap versus membership advantage, aggregated over all models and settings, for the two sampling techniques. Each row corresponds to a different attacker strength parameter $r \in \{0.1, 0.2, 0.5, 0.8\}$, representing the fraction of member and non-member nodes whose true membership labels are revealed to the attack model during training. Points in these plots are grouped by edge-access setting (Original Split, Full Graph, and No Edges). Each point shows the mean membership advantage at a given mean performance gap, while horizontal and vertical error bars denote one standard deviation in performance gap and membership advantage, respectively. We provide detailed performance-gap and membership-advantage scores for all datasets in Tables 9–16 (Cora), 17–24 (Chameleon), and 25–32 (PubMed) in Appendix C.

We observe that across all values of r, the relative ordering of the different graph-access settings and the overall relationship between performance gap and membership advantage remain largely stable, with only isolated exceptions. In addition, membership advantage tends to increase with r, especially when viewed in aggregate across settings. We therefore provide the following analysis without referring to any specific value of r.

First, from the scatter plots, we observe that in terms of membership advantage the original split setting dominates. Specifically, we observe that even when No Edges setting exhibits a comparable or even larger performance gap than the Original Split setting, it nevertheless leads to lower membership leakage.

Second, from the detailed scores we observe that in most cases, particularly for Gcn and Gat, moving from the Original Split to the Full Graph leads to a sharp drop in membership advantage. This trend is observed on both Cora and Chameleon, suggesting that access to the full graph often makes member and non-member nodes less distinguishable. The train and test accuracies provide further insight into this effect. On Cora, using all graph edges increases test accuracy, while on Chameleon, using all graph edges causes a substantial decrease in train accuracy and, in most cases, a small increase in test accuracy, thereby reducing the performance gap. This is consistent with the fact that Chameleon is a low-homophily graph, where adding more edges can degrade performance. Even so, message passing still creates a mixing effect that blurs the distinction between member and non-member nodes.

Third, the differences in membership advantage across different edge-access settings are often smaller for GraphSage. We hypothesize that this stems from GraphSage's separate treatment of self-node and neighborhood information, which can preserve more node-specific signal and make GraphSage less sensitive to biased neighbor sets than Gcn or Gat, which more directly mix node and neighbor messages.

Fourth, we discuss the special case of PubMed. Across both train-set sizes and sampling strategies, train and test accuracies remain relatively high, and the resulting membership advantage is comparatively low. This is broadly consistent with prior observations in the literature regarding the relative difficulty of attacking this dataset. A notable departure from the pattern seen in Cora and Chameleon concerns the relationship between performance gap and membership advantage in the FullGraph setting. On PubMed, using the full graph does not suppress leakage as consistently as it does on Cora. In particular, for Gcn under Snowball sampling,

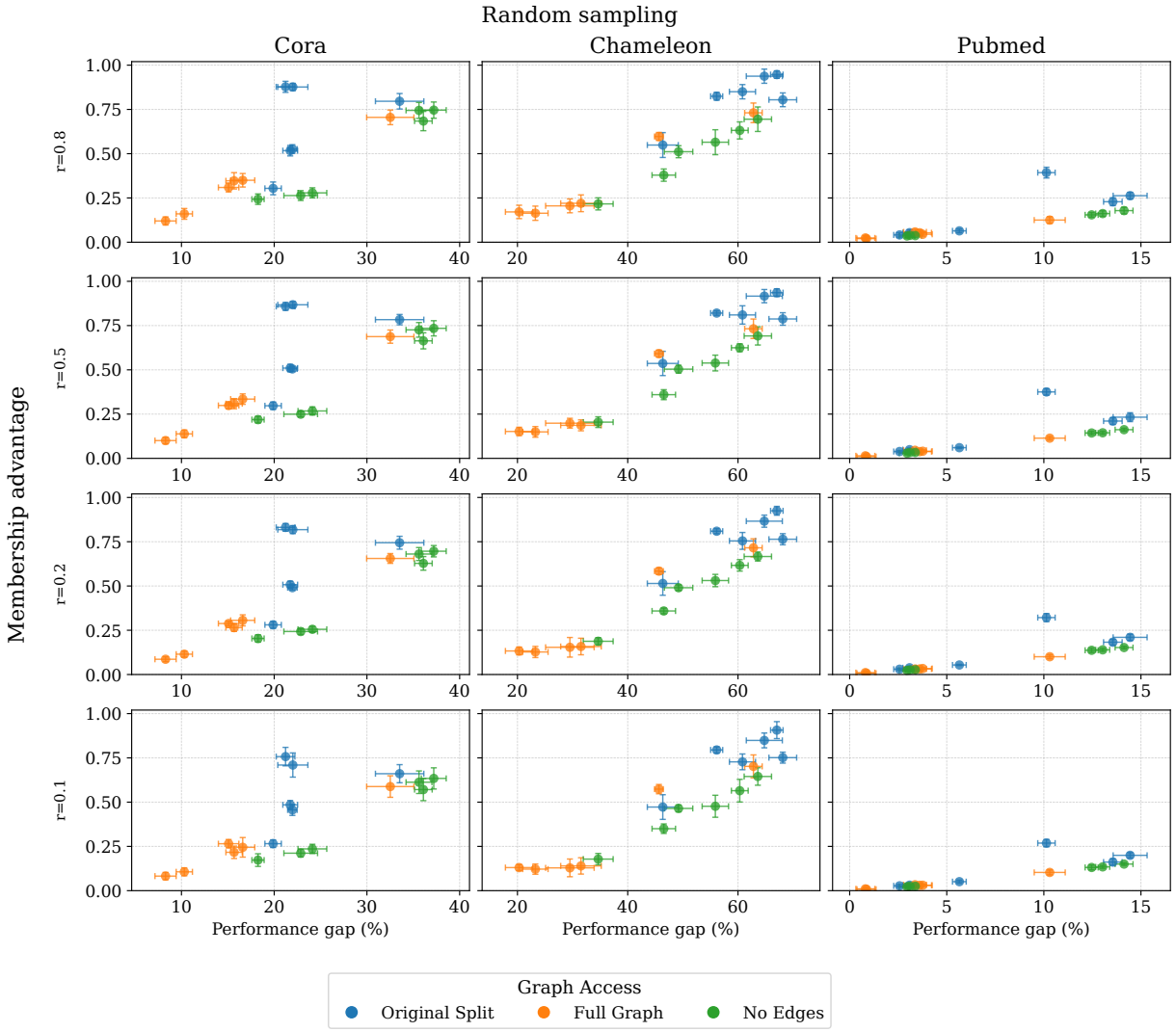

Figure 7: Scatter plot of performance gap versus membership advantage under random sampling shown for different attack-model training fractions $r \in \{0.1, 0.2, 0.5, 0.8\}$. Each row corresponds to one value of r, and points are grouped by edge-access setting (Original Split, Full Graph, and No Edges). Each point shows the mean membership advantage at a given mean performance gap, while horizontal and vertical error bars denote one standard deviation in performance gap and membership advantage, respectively.

FullGraph setting yields membership advantage that is comparable to, and in some cases higher than, that of the Original setting, even though the corresponding performance gap is lower.

Last, when comparing snowball and random sampling, we generally observe higher membership advantage under snowball sampling. This effect is most pronounced on PubMed, where, despite the overall membership advantage being relatively low, snowball sampling still consistently exhibits higher membership advantage compared to random sampling. A likely explanation is the coverage bias introduced by snowball sampling, which concentrates the sampled training graph within specific structural regions and repeatedly exposes the model to similar neighborhood patterns, thereby increasing behavioral differences between member and non-member nodes.

One important subtlety we observe is that for the homophilic graphs Cora and PubMed, the FullGraph setting can exhibit higher membership advantage than the NoEdges setting, even when the former achieves a lower performance gap. This behavior contrasts with the random sampling setting, where FullGraph often leads to the lowest membership advantage.

Summary. Overall our experimental results point to the fact that the edge structure is a crucial variable for understanding and quantifying privacy leakage. Alone performance or generalization gap does not suffice to explain the information leakage in graph based models. For realistic evaluation of privacy leakage in GNNs

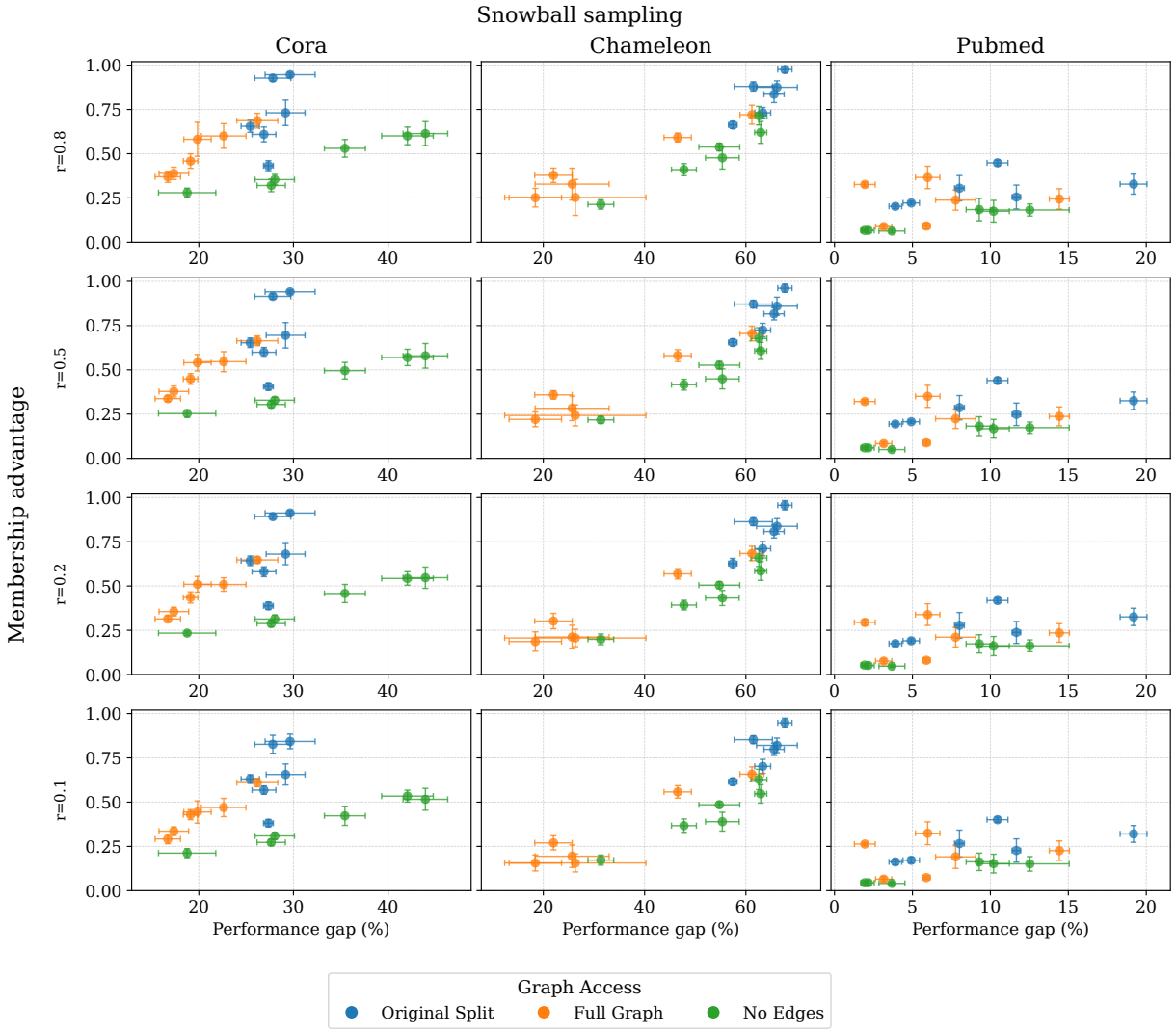

Figure 8: Scatter plot of performance gap versus membership advantage under snowball sampling shown for different attack-model training fractions $r \in \{0.1, 0.2, 0.5, 0.8\}$. Points and error bars have the same meaning as in the corresponding plot for random sampling.

we need to devise new and realistic assumptions on the available graph structure to the adversary. To develop more intelligent attacks new edge manipulation strategies would need to be developed that can help attacker to behave differently on member and non-member query nodes.

7 Related Work

Membership inference (MI) is a fundamental privacy risk for machine learning: black-box and white-box attacks exploit systematic differences in a model's behavior on training (member) vs. held-out (non-member) data [2, 11, 12, 26, 30, 36]. Recent work extends MI to graph neural networks (GNNs) by defining the *instance* at the node, edge, or graph level and tailoring the attack surface accordingly [4, 9, 21, 32–34]. Typical node-level attacks query a trained GNN with a target node (and optionally its neighborhood), using confidence scores, losses, or shadow-model posteriors as membership signals; threat models vary in (i) access to outputs/parameters and (ii) the relational context included with the query (features only vs. features+edges).

Despite this progress, the use of *relational structure* that distinguishes GNNs from i.i.d. models has not been examined systematically as a driver of MI risk. In particular, prior attacks typically treat the graph as a fixed backdrop and do not analyze how training graph construction and inference time edge access modulates attack success.

To defend against information leaks via membership inference with theoretical gurantees, *differential privacy* (DP) [6] has emerged as the most widely used formalism: it guarantees that the presence or absence of any single data point has only a limited effect on a

mechanism's output, thereby constraining an adversary's ability to infer membership. Existing methods for privacy preserving training of GNNs in centralised setting can be grouped into three main classes: (i) knowledge distillation based [22] in which the model trained on private data is never released but is used to train a public model under a weak supervision setting [20], (ii) gradient perturbation techniques in which typically DP-SGD [1] is extended for the case of graphs [35], and (iii) aggregation perturbation where the key idea is to add noise to the aggregate information obtained from the GNN neighborhood aggregation step [24, 25].

In i.i.d. domains, membership–inference (MI) attacks are widely used to *audit* differentially private (DP) learning by providing empirical lower bounds on leakage [12–14, 36]. While it may seem that these auditing algorithms carry over directly to graphs, we showed that this intuition can be misleading. We made the assumptions explicit and provided *theoretical* support for how train graph construction strategies affect the validity of these audits.

8 Discussion and Key Insights

We investigated node-level membership inference (MI) against GNNs in the inductive setting, formalizing the problem through a joint distribution over node–neighborhood tuples and an explicit membership inference experiment. Our analysis is grounded in a key distinction between graph learning and classical non-connected data settings: in GNNs, understanding privacy risk fundamentally requires understanding how the training graph is constructed, what structural properties it exhibits, and how graph structure is utilized during training and inference. Our theoretical and empirical results lead to the following key insights, which we hope will motivate future work in this direction.

8.1 Generalization Gap in GNNs

In machine learning, the generalization gap typically refers to the difference between empirical performance on the training set and expected performance on unseen samples drawn from the same underlying data distribution. A large gap is often associated with overfitting and increased memorization of training samples.

In graph-based node-level learning, however, the relationship between generalization gap and membership inference risk becomes more subtle because node instances are not processed independently. In GNNs, predictions depend not only on node features, but also on the neighborhood structure available during training and inference. Consequently, the generalization gap may depend not only on which nodes belong to the training set and the underlying node feature distribution, but also on which edges and neighborhood connections are exposed during training and inference.

Our empirical analysis confirms this subtlety. We observe that changing only the exposed edge structure can substantially alter the performance gap, while its relationship to membership advantage does not always follow the expected trend. In particular, when no edges are exposed during inference, the performance gap may increase relative to the Original split setting, whereas the membership advantage simultaneously decreases. Such observations suggest that classical notions of overfitting and memorization may need to be revisited in graph learning settings, since differences in model behavior can arise not only from learned parameters, but also from the structural context available during inference.

8.2 Consequences for Differentially Private Models.

We showed that train and test instances are not statistically exchangeable in practical graph learning settings (c.f. Theorem 3.2). As a result, the empirical membership advantage computed using membership inference attacks, including the experiment formalized in this work, cannot in general be interpreted as a lower bound on the differential privacy (DP) budget in graph settings.

To understand the above in more practical terms, consider first a strong adversary with access to the sampling strategy and $n-1$ data points. In such cases, the remaining node may become distinguishable purely because of structural constraints induced by the graph construction process. In particular, as shown in the proof of Theorem 3.2, certain node–neighborhood configurations may become impossible under specific sampling strategies and observed graph structure, allowing membership to be inferred independently of the learned model parameters, even when the model itself is trained using DP.

Furthermore, even for weaker adversaries without access to the remaining $n-1$ data points, graph-dependent sampling strategies can still influence the learning process through uneven graph coverage and under-representation of particular structural regions or node types. Such coverage bias can affect model generalization and increase behavioral differences between member and non-member nodes. Importantly, this distinguishability may arise even when the learning algorithm itself satisfies differential privacy, because part of the leakage originates from structural biases introduced by the graph construction and sampling process. Consequently, the observed membership advantage may not be fully explained or bounded by the DP parameters of the trained model alone. We therefore advocate the need for new privacy notions in graph machine learning that explicitly account for the training graph construction strategy.

8.3 Modeling Privacy Risk

As final remarks, our analysis suggests that both privacy attacks and defenses in graph learning require more realistic threat formulations. An important open question is what can meaningfully be protected once the graph construction or sampling mechanism itself is even partially known to the adversary. Moreover, since node properties themselves are influenced by the surrounding graph structure, different nodes cannot always be treated as equivalent from a privacy perspective. Nodes occupying different structural roles may inherently exhibit different levels of membership risk, requiring more fine-grained analysis of privacy leakage in graph learning settings.

Acknowledgments

This publication is part of the project PriXAI with file number VI.Vidi.243.224 of the research programme Vidi ENW which is financed by the Dutch Research Council (NWO) under the grant https://doi.org/10.61686/FATII77836. The author used generative

AI-based tools to revise parts of the text, improve flow and correct any typos, grammatical errors, and awkward phrasing.

References

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep Learning with Differential Privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 308–318. https://doi.org/10.1145/2976749.2978318

[2] Nicholas Carlini, Steve Chien, Milad Nasr, Shuang Song, Andreas Terzis, and Florian Tramer. 2022. Membership inference attacks from first principles. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1897–1914.

[3] Mauro Conti, Jiaxin Li, Stjepan Picek, and Jing Xu. 2022. Label-only membership inference attack against node-level graph neural networks. In *Proceedings of the 15th ACM Workshop on Artificial Intelligence and Security*. 1–12.

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

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

[6] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*. Springer, 265–284.

[7] Thomas Gaudelet, Ben Day, Arian R Jamasb, Jyothish Soman, Cristian Regep, Gertrude Liu, Jeremy BR Hayter, Richard Vickers, Charles Roberts, Jian Tang, et al. 2021. Utilizing graph machine learning within drug discovery and development. *Briefings in bioinformatics* 22, 6 (2021), bbab159.

[8] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).

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

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

[11] Hongsheng Hu, Zoran Salcic, Lichao Sun, Gillian Dobbie, Philip S Yu, and Xuyun Zhang. 2022. Membership inference attacks on machine learning: A survey. *ACM Computing Surveys (CSUR)* 54, 11s (2022), 1–37.

[12] Thomas Humphries, Simon Oya, Lindsey Tulloch, Matthew Rafuse, Ian Goldberg, Urs Hengartner, and Florian Kerschbaum. 2023. Investigating membership inference attacks under data dependencies. In *2023 IEEE 36th Computer Security Foundations Symposium (CSF)*. IEEE, 473–488.

[13] Matthew Jagielski, Jonathan Ullman, and Alina Oprea. 2020. Auditing differentially private machine learning: How private is private sgd? *Advances in Neural Information Processing Systems* 33 (2020), 22205–22216.

[14] Bargav Jayaraman and David Evans. 2019. Evaluating differentially private machine learning in practice. In *28th USENIX security symposium (USENIX security 19)*. 1895–1912.

[15] Abdellah Jnaini, Afafe Bettar, and Mohammed Amine Koulali. 2022. How Powerful are Membership Inference Attacks on Graph Neural Networks?. In *Proceedings of the 34th International Conference on Scientific and Statistical Database Management*. 1–4.

[16] Mishaal Kazmi, Hadrien Lautraite, Alireza Akbari, Qiaoyue Tang, Mauricio Soroco, Tao Wang, Sébastien Gambs, and Mathias Lécuyer. 2024. Panoramia: Privacy auditing of machine learning models without retraining. *Advances in Neural Information Processing Systems* 37 (2024), 57262–57300.

[17] Lee Kennedy-Shaffer, Xueting Qiu, and William P Hanage. 2021. Snowball sampling study design for serosurveys early in disease outbreaks. *American Journal of Epidemiology* 190, 9 (2021), 1918–1927.

[18] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *5th International Conference on Learning Representations, ICLR 2017*.

[19] Alan Mislove, Massimiliano Marcon, Krishna P Gummadi, Peter Druschel, and Bobby Bhattacharjee. 2007. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. 29–42.

[20] Iyiola E Olatunji, Thorben Funke, and Megha Khosla. 2023. Releasing graph neural networks with differential privacy guarantees. *Transactions on Machine Learning Research* (2023).

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

[22] Nicolas Papernot, Shuang Song, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Úlfar Erlingsson. 2018. Scalable private learning with pate. *arXiv preprint arXiv:1802.08908* (2018).

[23] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. 2019. Multi-scale Attributed Node Embedding. arXiv:1909.13021 [cs.LG]

[24] Sina Sajadmanesh and Daniel Gatica-Perez. 2021. Locally private graph neural networks. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2130–2145.

[25] Sina Sajadmanesh, Ali Shahin Shamsabadi, Aurélien Bellet, and Daniel Gatica-Perez. 2022. GAP: Differentially Private Graph Neural Networks with Aggregation Perturbation. *arXiv preprint arXiv:2203.00949* (2022).

[26] Ahmed Salem, Yang Zhang, Mathias Humbert, Pascal Berrang, Mario Fritz, and Michael Backes. 2018. Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models. *arXiv preprint arXiv:1806.01246* (2018).

[27] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. 2020. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*. PMLR, 8459–8468.

[28] Roman Schulte-Sasse, Stefan Budach, Denes Hnisz, and Annalisa Marsico. 2021. Integration of multiomics data with graph convolutional networks to identify new cancer genes and their associated molecular mechanisms. *Nature Machine Intelligence* 3, 6 (2021), 513–526.

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

[30] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*. IEEE, 3–18.

[31] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[32] Xiuling Wang and Wendy Hui Wang. 2024. Subgraph Structure Membership Inference Attacks against Graph Neural Networks. *Proceedings on Privacy Enhancing Technologies* (2024).

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

[34] Bang Wu, Xiangwen Yang, Shirui Pan, and Xingliang Yuan. 2021. Adapting membership inference attacks to GNN for graph classification: Approaches and implications. In *2021 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1421–1426.

[35] Zihang Xiang, Tianhao Wang, and Di Wang. 2024. Preserving node-level privacy in graph neural networks. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 4714–4732.

[36] Samuel Yeom, Irene Giacomelli, Matt Fredrikson, and Somesh Jha. 2018. Privacy risk in machine learning: Analyzing the connection to overfitting. In *2018 IEEE 31st computer security foundations symposium (CSF)*. IEEE, 268–282.

A Aggregation Operations in GNNs

We focus on three key aggregation operations in GNNs, which serve as the foundation for more complex aggregation methods. The first aggregation operation is based on the Graph Convolutional Network (Gcn) model proposed by [18]. In this model, each node's representation is updated at each layer by computing a weighted average of its own features and those of its neighbors. Let $\mathbf{d}$ be the graph degree vector obtained after adding self loops to all nodes. The ith element $\mathbf{d}_i$ denotes the degree of node i. The aggregation operation in Gcn is then given as

$$\mathbf{z}_i^{(\ell)} \leftarrow \mathbf{W}^{(\ell)} \sum_{j \in \mathcal{N}(i) \cup i} \frac{1}{\sqrt{\mathbf{d}_i \mathbf{d}_j}} \mathbf{x}_j^{(\ell-1)}. \tag{7}$$

Here $\mathbf{W}^{(\ell)}$ is the projection matrix corresponding to layer ℓ. Unlike Gcn, GraphSage [8] explicitly differentiates between the representation of the query node (referred to as the ego node) and that of its neighbors. In GraphSage, aggregation is performed by concatenating the ego node's representation with the aggregated representations of its neighbors. Several methods for neighbor aggregation are proposed; in this work, we adopt the mean aggregation method,

which computes the average of the neighbors' representations. After applying the projection operation, the simplified aggregation in GraphSage can be expressed as follows:

$$\mathbf{z}^{(\ell)} = \mathbf{W}_1^{(\ell)}\mathbf{x}_i^{(\ell-1)} + \mathbf{W}_2^{(\ell)} \cdot \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} \mathbf{x}_j^{(\ell-1)} \tag{8}$$

In both Gcn and GraphSage, the transformation step following aggregation is simply the application of a ReLU non-linearity to the aggregated representation. Gat [31] introduces attention weights over the edges to perform the aggregation operation as follows.

$$\mathbf{z}_i^{(\ell,p)} = \sum_{j \in \mathcal{N}(i) \cup i} \alpha_{i,j}^{(p)} \mathbf{W}^{(\ell,p)} \mathbf{x}_j^{(\ell-1)}, \tag{9}$$

where the attention coefficients $\alpha_{i,j}^{(p)} : (\mathbf{x}_i, \mathbf{x}_j) \rightarrow \mathbb{R}$ can be seen as the importance weights for aggregation over edge (i, j) and computed as in [31]. In the transformation step the P representations corresponding to P attention mechanisms are concatenated after a non-linear transformation to obtain a single representation at layer ℓ.

$$\mathbf{x}_i^{(\ell)} = ||_{p=1}^{P} \text{ReLU}\left(\mathbf{z}_i^{(\ell,p)} \mathbf{W}^{(p\ell)}\right). \tag{10}$$

In this work we only experiment with the above representative aggregation types and do not account for other differences in these methods. For example, GraphSage was proposed to improve scalability in GNN training in which neighborhood sampling was used to reduce the size of the computational graph. In our implementation we restrict to only vary the aggregation types and the provided neighborhood of the input query node is completely used.

B Hyperparameter Details

All three target models were implemented as two-layer graph neural networks with a hidden dimension of 64 and were trained for 200 epochs using the Adam optimizer with learning rate 0.01 and weight decay 5×10^{-4}. For Gat, we used 8 attention heads in the first layer and a single attention head in the output layer.

The membership inference attack model was implemented as a 2-layer MLP with a hidden layer of dimension 64 and a single output unit. Each of the attack models was trained for 200 epochs using Adam with learning rate 0.001 and batch size 32. Binary cross-entropy loss was applied to sigmoid-transformed outputs. No dropout or additional hyperparameter tuning was used in the current implementation.

C Detailed Results

Detailed train and test accuracy scores for all datasets are presented in Tables 3–4 (Cora), 5–6 (Chameleon), and 7–8 (PubMed). We further provide detailed performance-gap and membership-advantage scores for all datasets in Tables 9–16 (Cora), 17–24 (Chameleon), and 25–32 (PubMed).

Table 3: Performance on Cora with 10% of nodes in the training set of the victim model under random and snowball sampling.

Model	Metric	Random			Snowball		
		Original split	Full graph	No edges	Original split	Full graph	No edges
GCN	Train Acc	0.9874 ± 0.0083	0.9578 ± 0.0127	0.9896 ± 0.0064	0.9963 ± 0.0023	0.9615 ± 0.0095	0.9459 ± 0.0093
	Test Acc	0.7779 ± 0.0136	0.7987 ± 0.0136	0.6214 ± 0.0144	0.7057 ± 0.0210	0.7438 ± 0.0191	0.5302 ± 0.0224
GraphSAGE	Train Acc	1.0000 ± 0.0000	0.9985 ± 0.0030	0.9807 ± 0.0059	1.0000 ± 0.0000	0.9956 ± 0.0028	0.9319 ± 0.0268
	Test Acc	0.6647 ± 0.0261	0.6738 ± 0.0261	0.6268 ± 0.0110	0.7035 ± 0.0264	0.7349 ± 0.0227	0.5400 ± 0.0303
GAT	Train Acc	0.9867 ± 0.0093	0.9356 ± 0.0160	0.9800 ± 0.0038	0.9911 ± 0.0055	0.9296 ± 0.0189	0.8911 ± 0.0138
	Test Acc	0.7696 ± 0.0200	0.7889 ± 0.0164	0.6310 ± 0.0155	0.7153 ± 0.0190	0.7450 ± 0.0213	0.5751 ± 0.0166

Table 4: Performance on Cora with 50% of nodes in the training set of the victim model under random and snowball sampling.

Model	Metric	Random			Snowball		
		Original split	Full graph	No edges	Original split	Full graph	No edges
GCN	Train Acc	0.9880 ± 0.0022	0.9499 ± 0.0058	0.8954 ± 0.0050	0.9968 ± 0.0013	0.9694 ± 0.0049	0.8471 ± 0.0044
	Test Acc	0.7916 ± 0.0088	0.8712 ± 0.0090	0.6795 ± 0.0151	0.7241 ± 0.0054	0.8072 ± 0.0098	0.6129 ± 0.0125
GraphSAGE	Train Acc	1.0000 ± 0.0000	0.9929 ± 0.0027	0.8913 ± 0.0158	1.0000 ± 0.0000	0.9985 ± 0.0014	0.8781 ± 0.0097
	Test Acc	0.7827 ± 0.0079	0.8431 ± 0.0111	0.6873 ± 0.0090	0.7456 ± 0.0096	0.8075 ± 0.0075	0.6319 ± 0.0156
GAT	Train Acc	0.9886 ± 0.0028	0.9334 ± 0.0062	0.8487 ± 0.0066	0.9956 ± 0.0019	0.9535 ± 0.0053	0.8007 ± 0.0091
	Test Acc	0.7715 ± 0.0062	0.8372 ± 0.0123	0.6938 ± 0.0079	0.7279 ± 0.0124	0.7879 ± 0.0113	0.6502 ± 0.0179

Table 5: Performance on Chameleon with 10% of nodes in the training set of the victim model under random and snowball sampling.

Model	Metric	Random			Snowball		
		Original split	Full graph	No edges	Original split	Full graph	No edges
GCN	Train Acc	0.9454 ± 0.0132	0.5683 ± 0.0251	0.8626 ± 0.0188	0.9912 ± 0.0074	0.5366 ± 0.0522	0.8432 ± 0.0389
	Test Acc	0.3706 ± 0.0252	0.4001 ± 0.0236	0.3421 ± 0.0126	0.3414 ± 0.0205	0.3952 ± 0.0180	0.3125 ± 0.0155
GraphSAGE	Train Acc	0.9921 ± 0.0051	0.9330 ± 0.0329	0.8520 ± 0.0132	0.9956 ± 0.0039	0.9383 ± 0.0169	0.8634 ± 0.0572
	Test Acc	0.3269 ± 0.0103	0.3468 ± 0.0134	0.3100 ± 0.0205	0.3216 ± 0.0137	0.3643 ± 0.0243	0.3230 ± 0.0238
GAT	Train Acc	0.9445 ± 0.0117	0.5410 ± 0.0660	0.7463 ± 0.0259	0.9639 ± 0.0085	0.3894 ± 0.0828	0.6643 ± 0.0525
	Test Acc	0.3329 ± 0.0327	0.3696 ± 0.0429	0.3296 ± 0.0266	0.3264 ± 0.0392	0.2799 ± 0.0589	0.2959 ± 0.0244

Table 6: Performance on Chameleon with 50% of nodes in the training set of the victim model under random and snowball sampling.

Model	Metric	Random			Snowball		
		Original split	Full graph	No edges	Original split	Full graph	No edges
GCN	Train Acc	0.8629 ± 0.0106	0.7409 ± 0.0156	0.7330 ± 0.0131	0.9661 ± 0.0040	0.6455 ± 0.0081	0.7418 ± 0.0136
	Test Acc	0.4627 ± 0.0209	0.5905 ± 0.0210	0.3917 ± 0.0136	0.4114 ± 0.0091	0.5034 ± 0.0229	0.3874 ± 0.0167
GraphSAGE	Train Acc	0.9960 ± 0.0027	0.9668 ± 0.0092	0.8062 ± 0.0279	0.9998 ± 0.0004	0.9649 ± 0.0069	0.8236 ± 0.0267
	Test Acc	0.4369 ± 0.0102	0.5254 ± 0.0085	0.4093 ± 0.0245	0.3853 ± 0.0377	0.5157 ± 0.0225	0.3731 ± 0.0416
GAT	Train Acc	0.9388 ± 0.0084	0.6480 ± 0.0535	0.5227 ± 0.0140	0.9757 ± 0.0117	0.4982 ± 0.0299	0.4996 ± 0.0232
	Test Acc	0.2994 ± 0.0252	0.4973 ± 0.0434	0.3415 ± 0.0158	0.3579 ± 0.0153	0.4063 ± 0.0340	0.3426 ± 0.0124

Table 7: Performance on PubMed with 10% of nodes in the training set of the victim model under random and snowball sampling.

Model	Metric	Random			Snowball		
		Original split	Full graph	No edges	Original split	Full graph	No edges
GCN	Train Acc	0.9559 ± 0.0042	0.8920 ± 0.0054	0.9526 ± 0.0023	0.9081 ± 0.0037	0.8988 ± 0.0077	0.8856 ± 0.0069
	Test Acc	0.8590 ± 0.0029	0.8620 ± 0.0030	0.8285 ± 0.0047	0.8353 ± 0.0027	0.8450 ± 0.0024	0.8034 ± 0.0049
GraphSAGE	Train Acc	0.9812 ± 0.0019	0.9371 ± 0.0058	0.9505 ± 0.0036	0.9943 ± 0.0030	0.9643 ± 0.0072	0.8202 ± 0.0266
	Test Acc	0.8395 ± 0.0091	0.8405 ± 0.0089	0.8320 ± 0.0034	0.8036 ± 0.0104	0.8252 ± 0.0111	0.7172 ± 0.0254
GAT	Train Acc	0.9651 ± 0.0035	0.8674 ± 0.0084	0.9562 ± 0.0030	0.9195 ± 0.0081	0.8918 ± 0.0134	0.8675 ± 0.0080
	Test Acc	0.8343 ± 0.0053	0.8361 ± 0.0051	0.8211 ± 0.0057	0.8123 ± 0.0086	0.8224 ± 0.0095	0.7789 ± 0.0077

Table 8: Performance on PubMed with 50% of nodes in the training set of the victim model under random and snowball sampling.

Model	Metric	Random			Snowball		
		Original split	Full graph	No edges	Original split	Full graph	No edges
GCN	Train Acc	0.8920 ± 0.0010	0.8845 ± 0.0023	0.8875 ± 0.0030	0.8872 ± 0.0024	0.8885 ± 0.0038	0.8647 ± 0.0020
	Test Acc	0.8691 ± 0.0024	0.8768 ± 0.0023	0.8613 ± 0.0025	0.8526 ± 0.0027	0.8712 ± 0.0023	0.8480 ± 0.0011
GraphSAGE	Train Acc	0.9277 ± 0.0023	0.9200 ± 0.0026	0.8980 ± 0.0017	0.9337 ± 0.0030	0.9317 ± 0.0025	0.8555 ± 0.0136
	Test Acc	0.8753 ± 0.0027	0.8854 ± 0.0028	0.8677 ± 0.0015	0.8361 ± 0.0072	0.8768 ± 0.0022	0.8239 ± 0.0080
GAT	Train Acc	0.8897 ± 0.0010	0.8741 ± 0.0029	0.8873 ± 0.0016	0.8875 ± 0.0033	0.8869 ± 0.0037	0.8590 ± 0.0034
	Test Acc	0.8623 ± 0.0034	0.8670 ± 0.0027	0.8598 ± 0.0013	0.8438 ± 0.0021	0.8589 ± 0.0012	0.8404 ± 0.0028

Table 9: Performance gap (in %) and membership advantage on Cora when 10% of nodes are in the training set of the target model and the attacker train fraction is $r = 0.1$.

Model	**Metric**	**Random**			**Snowball**		
		Original split	**Full graph**	**No edges**	**Original split**	**Full graph**	**No edges**
GCN	Perf. Gap	21.2257 ± 1.0071	16.6048 ± 1.2985	37.2089 ± 1.3402	29.1732 ± 2.0505	22.6257 ± 2.3590	43.9479 ± 2.3569
	Adv.	0.7572 ± 0.0521	0.2448 ± 0.0552	0.6340 ± 0.0598	0.6564 ± 0.0594	0.4697 ± 0.0515	0.5159 ± 0.0616
GraphSage	Perf. Gap	33.5275 ± 2.6085	32.5189 ± 2.5481	36.0882 ± 0.9495	29.6473 ± 2.6414	26.1806 ± 2.1638	42.0500 ± 2.7268
	Adv.	0.6606 ± 0.0511	0.5880 ± 0.0607	0.5709 ± 0.0625	0.8430 ± 0.0413	0.6106 ± 0.0234	0.5340 ± 0.0337
GAT	Perf. Gap	22.0094 ± 1.6211	15.6743 ± 0.8820	35.6151 ± 1.3949	27.8237 ± 1.9008	19.8598 ± 1.4470	35.4403 ± 2.1669
	Adv.	0.7096 ± 0.0682	0.2167 ± 0.0350	0.6125 ± 0.0637	0.8268 ± 0.0509	0.4441 ± 0.0628	0.4228 ± 0.0540

Table 10: Performance gap (in %) and membership advantage on Cora when 10% of nodes are in the training set of the target model and the attacker train fraction is $r = 0.2$.

Model	**Metric**	**Random**			**Snowball**		
		Original split	**Full graph**	**No edges**	**Original split**	**Full graph**	**No edges**
GCN	Perf. Gap	21.2257 ± 1.0071	16.6048 ± 1.2985	37.2089 ± 1.3402	29.1732 ± 2.0505	22.6257 ± 2.3590	43.9479 ± 2.3569
	Adv.	0.8310 ± 0.0217	0.3059 ± 0.0312	0.6964 ± 0.0320	0.6801 ± 0.0598	0.5085 ± 0.0379	0.5470 ± 0.0602
GraphSage	Perf. Gap	33.5275 ± 2.6085	32.5189 ± 2.5481	36.0882 ± 0.9495	29.6473 ± 2.6414	26.1806 ± 2.1638	42.0500 ± 2.7268
	Adv.	0.7443 ± 0.0363	0.6555 ± 0.0276	0.6278 ± 0.0385	0.9122 ± 0.0128	0.6472 ± 0.0175	0.5426 ± 0.0381
GAT	Perf. Gap	22.0094 ± 1.6211	15.6743 ± 0.8820	35.6151 ± 1.3949	27.8237 ± 1.9008	19.8598 ± 1.4470	35.4403 ± 2.1669
	Adv.	0.8183 ± 0.0225	0.2648 ± 0.0214	0.6805 ± 0.0365	0.8918 ± 0.0141	0.5096 ± 0.0446	0.4576 ± 0.0514

Table 11: Performance gap (in %) and membership advantage on Cora when 10% of nodes are in the training set of the target model and the attacker train fraction is $r = 0.5$.

Model	Metric	Random			Snowball		
		Original split	Full graph	No edges	Original split	Full graph	No edges
GCN	Perf. Gap	21.2257 ± 1.0071	16.6048 ± 1.2985	37.2089 ± 1.3402	29.1732 ± 2.0505	22.6257 ± 2.3590	43.9479 ± 2.3569
	Adv.	0.8581 ± 0.0222	0.3347 ± 0.0294	0.7345 ± 0.0430	0.6951 ± 0.0718	0.5461 ± 0.0565	0.5792 ± 0.0696
GraphSage	Perf. Gap	33.5275 ± 2.6085	32.5189 ± 2.5481	36.0882 ± 0.9495	29.6473 ± 2.6414	26.1806 ± 2.1638	42.0500 ± 2.7268
	Adv.	0.7835 ± 0.0292	0.6876 ± 0.0373	0.6638 ± 0.0454	0.9411 ± 0.0110	0.6642 ± 0.0278	0.5700 ± 0.0461
GAT	Perf. Gap	22.0094 ± 1.6211	15.6743 ± 0.8820	35.6151 ± 1.3949	27.8237 ± 1.9008	19.8598 ± 1.4470	35.4403 ± 2.1669
	Adv.	0.8675 ± 0.0180	0.3088 ± 0.0286	0.7258 ± 0.0413	0.9154 ± 0.0152	0.5404 ± 0.0466	0.4954 ± 0.0468

Table 12: Performance gap (in %) and membership advantage on Cora when 10% of nodes are in the training set of the target model and the attacker train fraction is $r = 0.8$.

Model	Metric	Random			Snowball		
		Original split	Full graph	No edges	Original split	Full graph	No edges
GCN	Perf. Gap	21.2257 ± 1.0071	16.6048 ± 1.2985	37.2089 ± 1.3402	29.1732 ± 2.0505	22.6257 ± 2.3590	43.9479 ± 2.3569
	Adv.	0.8776 ± 0.0312	0.3498 ± 0.0381	0.7461 ± 0.0458	0.7309 ± 0.0720	0.6001 ± 0.0695	0.6136 ± 0.0674
GraphSage	Perf. Gap	33.5275 ± 2.6085	32.5189 ± 2.5481	36.0882 ± 0.9495	29.6473 ± 2.6414	26.1806 ± 2.1638	42.0500 ± 2.7268
	Adv.	0.7961 ± 0.0437	0.7052 ± 0.0413	0.6843 ± 0.0545	0.9462 ± 0.0127	0.6865 ± 0.0415	0.6006 ± 0.0499
GAT	Perf. Gap	22.0094 ± 1.6211	15.6743 ± 0.8820	35.6151 ± 1.3949	27.8237 ± 1.9008	19.8598 ± 1.4470	35.4403 ± 2.1669
	Adv.	0.8767 ± 0.0205	0.3468 ± 0.0461	0.7445 ± 0.0449	0.9270 ± 0.0172	0.5812 ± 0.0959	0.5302 ± 0.0497

Table 13: Performance gap (in %) and membership advantage on Cora when 50% of nodes are in the training set of the target model and the attacker train fraction is $r = 0.1$.

Model	Metric	Random			Snowball		
		Original split	Full graph	No edges	Original split	Full graph	No edges
GCN	Perf. Gap	19.8832 ± 0.8856	8.2840 ± 1.1413	24.1190 ± 1.5496	27.3564 ± 0.5142	16.7243 ± 1.3459	27.6532 ± 1.4966
	Adv.	0.2654 ± 0.0217	0.0822 ± 0.0209	0.2359 ± 0.0260	0.3814 ± 0.0204	0.2920 ± 0.0271	0.2726 ± 0.0172
GraphSage	Perf. Gap	21.7282 ± 0.7946	15.0847 ± 1.1030	22.8597 ± 1.8146	25.4357 ± 0.9648	19.1270 ± 0.7650	28.0278 ± 2.0823
	Adv.	0.4848 ± 0.0243	0.2651 ± 0.0238	0.2117 ± 0.0218	0.6307 ± 0.0245	0.4294 ± 0.0283	0.3091 ± 0.0194
GAT	Perf. Gap	21.9638 ± 0.5095	10.3057 ± 0.8758	18.2564 ± 0.6569	26.8838 ± 1.2589	17.3588 ± 1.5653	18.7669 ± 3.0288
	Adv.	0.4561 ± 0.0312	0.1064 ± 0.0226	0.1727 ± 0.0357	0.5680 ± 0.0237	0.3360 ± 0.0244	0.2119 ± 0.0254

Table 14: Performance gap (in %) and membership advantage on Cora when 50% of nodes are in the training set of the target model and the attacker train fraction is $r = 0.2$.

Model	Metric	Random			Snowball		
		Original split	Full graph	No edges	Original split	Full graph	No edges
GCN	Perf. Gap	19.8832 ± 0.8856	8.2840 ± 1.1413	24.1190 ± 1.5496	27.3564 ± 0.5142	16.7243 ± 1.3459	27.6532 ± 1.4966
	Adv.	0.2805 ± 0.0194	0.0862 ± 0.0136	0.2553 ± 0.0171	0.3879 ± 0.0210	0.3140 ± 0.0183	0.2877 ± 0.0177
GraphSage	Perf. Gap	21.7282 ± 0.7946	15.0847 ± 1.1030	22.8597 ± 1.8146	25.4357 ± 0.9648	19.1270 ± 0.7650	28.0278 ± 2.0823
	Adv.	0.5074 ± 0.0195	0.2880 ± 0.0127	0.2429 ± 0.0174	0.6426 ± 0.0272	0.4354 ± 0.0315	0.3135 ± 0.0229
GAT	Perf. Gap	21.9638 ± 0.5095	10.3057 ± 0.8758	18.2564 ± 0.6569	26.8838 ± 1.2589	17.3588 ± 1.5653	18.7669 ± 3.0288
	Adv.	0.4905 ± 0.0129	0.1148 ± 0.0170	0.2036 ± 0.0209	0.5806 ± 0.0278	0.3555 ± 0.0244	0.2336 ± 0.0121

Table 15: Performance gap (in %) and membership advantage on Cora when 50% of nodes are in the training set of the target model and the attacker train fraction is $r = 0.5$.

Model	Metric	Random			Snowball		
		Original split	Full graph	No edges	Original split	Full graph	No edges
GCN	Perf. Gap	19.8832 ± 0.8856	8.2840 ± 1.1413	24.1190 ± 1.5496	27.3564 ± 0.5142	16.7243 ± 1.3459	27.6532 ± 1.4966
	Adv.	0.2971 ± 0.0210	0.1004 ± 0.0183	0.2677 ± 0.0231	0.4056 ± 0.0209	0.3374 ± 0.0170	0.3045 ± 0.0188
GraphSage	Perf. Gap	21.7282 ± 0.7946	15.0847 ± 1.1030	22.8597 ± 1.8146	25.4357 ± 0.9648	19.1270 ± 0.7650	28.0278 ± 2.0823
	Adv.	0.5099 ± 0.0212	0.2987 ± 0.0194	0.2495 ± 0.0156	0.6530 ± 0.0269	0.4486 ± 0.0300	0.3285 ± 0.0168
GAT	Perf. Gap	21.9638 ± 0.5095	10.3057 ± 0.8758	18.2564 ± 0.6569	26.8838 ± 1.2589	17.3588 ± 1.5653	18.7669 ± 3.0288
	Adv.	0.5052 ± 0.0129	0.1383 ± 0.0222	0.2188 ± 0.0188	0.5993 ± 0.0273	0.3776 ± 0.0310	0.2529 ± 0.0208

Table 16: Performance gap (in %) and membership advantage on Cora when 50% of nodes are in the training set of the target model and the attacker train fraction is $r = 0.8$.

Model	Metric	Random			Snowball		
		Original split	Full graph	No edges	Original split	Full graph	No edges
GCN	Perf. Gap	19.8832 ± 0.8856	8.2840 ± 1.1413	24.1190 ± 1.5496	27.3564 ± 0.5142	16.7243 ± 1.3459	27.6532 ± 1.4966
	Adv.	0.3041 ± 0.0360	0.1200 ± 0.0230	0.2787 ± 0.0285	0.4325 ± 0.0285	0.3695 ± 0.0317	0.3205 ± 0.0353
GraphSage	Perf. Gap	21.7282 ± 0.7946	15.0847 ± 1.1030	22.8597 ± 1.8146	25.4357 ± 0.9648	19.1270 ± 0.7650	28.0278 ± 2.0823
	Adv.	0.5171 ± 0.0297	0.3092 ± 0.0251	0.2635 ± 0.0278	0.6549 ± 0.0338	0.4585 ± 0.0411	0.3535 ± 0.0294
GAT	Perf. Gap	21.9638 ± 0.5095	10.3057 ± 0.8758	18.2564 ± 0.6569	26.8838 ± 1.2589	17.3588 ± 1.5653	18.7669 ± 3.0288
	Adv.	0.5260 ± 0.0247	0.1601 ± 0.0303	0.2431 ± 0.0296	0.6089 ± 0.0419	0.3889 ± 0.0344	0.2797 ± 0.0256

Table 17: Performance gap (in %) and membership advantage on Chameleon when 10% of nodes are in the training set of the target model and the attacker train fraction is $r = 0.1$.

Model	Metric	Random			Snowball		
		Original split	Full graph	No edges	Original split	Full graph	No edges
GCN	Perf. Gap	60.8093 ± 2.3679	29.5093 ± 4.4010	60.3211 ± 1.5209	65.5633 ± 1.9914	25.6865 ± 7.3105	62.9266 ± 1.1975
	Adv.	0.7275 ± 0.0445	0.1287 ± 0.0500	0.5647 ± 0.0640	0.7986 ± 0.0347	0.1944 ± 0.0636	0.5467 ± 0.0521
GraphSage	Perf. Gap	67.0425 ± 1.1286	62.7990 ± 1.5861	63.5992 ± 2.4818	67.7003 ± 1.4069	61.1871 ± 2.3532	62.5690 ± 1.5817
	Adv.	0.9068 ± 0.0479	0.7020 ± 0.0647	0.6449 ± 0.0494	0.9483 ± 0.0258	0.6581 ± 0.0414	0.6280 ± 0.0576
GAT	Perf. Gap	64.7719 ± 3.2673	31.5193 ± 3.6788	55.8904 ± 2.3936	66.1328 ± 4.0300	26.3182 ± 13.9418	55.3452 ± 3.3219
	Adv.	0.8486 ± 0.0418	0.1398 ± 0.0461	0.4767 ± 0.0621	0.8209 ± 0.0411	0.1564 ± 0.0508	0.3897 ± 0.0530

Table 18: Performance gap (in %) and membership advantage on Chameleon when 10% of nodes are in the training set of the target model and the attacker train fraction is $r = 0.2$.

Model	Metric	Random			Snowball		
		Original split	Full graph	No edges	Original split	Full graph	No edges
GCN	Perf. Gap	60.8093 ± 2.3679	29.5093 ± 4.4010	60.3211 ± 1.5209	65.5633 ± 1.9914	25.6865 ± 7.3105	62.9266 ± 1.1975
	Adv.	0.7547 ± 0.0475	0.1537 ± 0.0554	0.6165 ± 0.0323	0.8078 ± 0.0366	0.2116 ± 0.0666	0.5845 ± 0.0524
GraphSage	Perf. Gap	67.0425 ± 1.1286	62.7990 ± 1.5861	63.5992 ± 2.4818	67.7003 ± 1.4069	61.1871 ± 2.3532	62.5690 ± 1.5817
	Adv.	0.9247 ± 0.0248	0.7156 ± 0.0509	0.6665 ± 0.0254	0.9561 ± 0.0262	0.6837 ± 0.0408	0.6580 ± 0.0478
GAT	Perf. Gap	64.7719 ± 3.2673	31.5193 ± 3.6788	55.8904 ± 2.3936	66.1328 ± 4.0300	26.3182 ± 13.9418	55.3452 ± 3.3219
	Adv.	0.8661 ± 0.0342	0.1582 ± 0.0467	0.5308 ± 0.0347	0.8371 ± 0.0432	0.2064 ± 0.0491	0.4321 ± 0.0426

Table 19: Performance gap (in %) and membership advantage on Chameleon when 10% of nodes are in the training set of the target model and the attacker train fraction is $r = 0.5$.

Model	Metric	Random			Snowball		
		Original split	Full graph	No edges	Original split	Full graph	No edges
GCN	Perf. Gap	60.8093 ± 2.3679	29.5093 ± 4.4010	60.3211 ± 1.5209	65.5633 ± 1.9914	25.6865 ± 7.3105	62.9266 ± 1.1975
	Adv.	0.8101 ± 0.0516	0.1986 ± 0.0276	0.6247 ± 0.0223	0.8162 ± 0.0341	0.2827 ± 0.0693	0.6073 ± 0.0481
GraphSage	Perf. Gap	67.0425 ± 1.1286	62.7990 ± 1.5861	63.5992 ± 2.4818	67.7003 ± 1.4069	61.1871 ± 2.3532	62.5690 ± 1.5817
	Adv.	0.9358 ± 0.0221	0.7320 ± 0.0550	0.6915 ± 0.0513	0.9611 ± 0.0226	0.7051 ± 0.0414	0.6774 ± 0.0607
GAT	Perf. Gap	64.7719 ± 3.2673	31.5193 ± 3.6788	55.8904 ± 2.3936	66.1328 ± 4.0300	26.3182 ± 13.9418	55.3452 ± 3.3219
	Adv.	0.9163 ± 0.0374	0.1866 ± 0.0308	0.5385 ± 0.0445	0.8592 ± 0.0513	0.2425 ± 0.0592	0.4490 ± 0.0568

Table 20: Performance gap (in %) and membership advantage on Chameleon when 10% of nodes are in the training set of the target model and the attacker train fraction is $r = 0.8$.

Model	Metric	Random			Snowball		
		Original split	Full graph	No edges	Original split	Full graph	No edges
GCN	Perf. Gap	60.8093 ± 2.3679	29.5093 ± 4.4010	60.3211 ± 1.5209	65.5633 ± 1.9914	25.6865 ± 7.3105	62.9266 ± 1.1975
	Adv.	0.8500 ± 0.0403	0.2053 ± 0.0391	0.6315 ± 0.0483	0.8360 ± 0.0472	0.3281 ± 0.0901	0.6202 ± 0.0616
GraphSage	Perf. Gap	67.0425 ± 1.1286	62.7990 ± 1.5861	63.5992 ± 2.4818	67.7003 ± 1.4069	61.1871 ± 2.3532	62.5690 ± 1.5817
	Adv.	0.9468 ± 0.0212	0.7308 ± 0.0553	0.6947 ± 0.0692	0.9753 ± 0.0182	0.7201 ± 0.0538	0.7145 ± 0.0525
GAT	Perf. Gap	64.7719 ± 3.2673	31.5193 ± 3.6788	55.8904 ± 2.3936	66.1328 ± 4.0300	26.3182 ± 13.9418	55.3452 ± 3.3219
	Adv.	0.9377 ± 0.0400	0.2199 ± 0.0473	0.5648 ± 0.0704	0.8752 ± 0.0362	0.2527 ± 0.1014	0.4767 ± 0.0635

Table 21: Performance gap (in %) and membership advantage on Chameleon when 50% of nodes are in the training set of the target model and the attacker train fraction is $r = 0.1$.

Model	Metric	Random			Snowball		
		Original split	Full graph	No edges	Original split	Full graph	No edges
GCN	Perf. Gap	46.3581 ± 2.8051	20.2978 ± 2.4804	46.5397 ± 2.1297	57.4159 ± 0.8470	22.0002 ± 3.6979	47.7655 ± 2.4608
	Adv.	0.4723 ± 0.0697	0.1303 ± 0.0169	0.3501 ± 0.0268	0.6154 ± 0.0202	0.2699 ± 0.0399	0.3672 ± 0.0378
GraphSage	Perf. Gap	56.1335 ± 1.0929	45.6566 ± 0.7884	49.2196 ± 2.5772	61.4682 ± 3.7748	46.5315 ± 2.6960	54.7559 ± 4.0392
	Adv.	0.7947 ± 0.0175	0.5735 ± 0.0262	0.4645 ± 0.0194	0.8529 ± 0.0222	0.5577 ± 0.0351	0.4853 ± 0.0166
GAT	Perf. Gap	68.1249 ± 2.5124	23.2490 ± 2.2998	34.6459 ± 2.7079	63.3213 ± 1.5763	18.4210 ± 5.1792	31.3578 ± 2.5447
	Adv.	0.7513 ± 0.0309	0.1217 ± 0.0289	0.1783 ± 0.0319	0.7021 ± 0.0395	0.1562 ± 0.0450	0.1724 ± 0.0285

Table 22: Performance gap (in %) and membership advantage on Chameleon when 50% of nodes are in the training set of the target model and the attacker train fraction is $r = 0.2$.

Model	Metric	Random			Snowball		
		Original split	Full graph	No edges	Original split	Full graph	No edges
GCN	Perf. Gap	46.3581 ± 2.8051	20.2978 ± 2.4804	46.5397 ± 2.1297	57.4159 ± 0.8470	22.0002 ± 3.6979	47.7655 ± 2.4608
	Adv.	0.5142 ± 0.0665	0.1330 ± 0.0221	0.3589 ± 0.0162	0.6264 ± 0.0296	0.3019 ± 0.0434	0.3920 ± 0.0271
GraphSage	Perf. Gap	56.1335 ± 1.0929	45.6566 ± 0.7884	49.2196 ± 2.5772	61.4682 ± 3.7748	46.5315 ± 2.6960	54.7559 ± 4.0392
	Adv.	0.8094 ± 0.0142	0.5839 ± 0.0185	0.4902 ± 0.0162	0.8630 ± 0.0219	0.5688 ± 0.0293	0.5047 ± 0.0196
GAT	Perf. Gap	68.1249 ± 2.5124	23.2490 ± 2.2998	34.6459 ± 2.7079	63.3213 ± 1.5763	18.4210 ± 5.1792	31.3578 ± 2.5447
	Adv.	0.7641 ± 0.0314	0.1273 ± 0.0316	0.1877 ± 0.0204	0.7109 ± 0.0405	0.1862 ± 0.0547	0.1988 ± 0.0302

Table 23: Performance gap (in %) and membership advantage on Chameleon when 50% of nodes are in the training set of the target model and the attacker train fraction is $r = 0.5$.

Model	Metric	Random			Snowball		
		Original split	Full graph	No edges	Original split	Full graph	No edges
GCN	Perf. Gap	46.3581 ± 2.8051	20.2978 ± 2.4804	46.5397 ± 2.1297	57.4159 ± 0.8470	22.0002 ± 3.6979	47.7655 ± 2.4608
	Adv.	0.5361 ± 0.0684	0.1519 ± 0.0237	0.3597 ± 0.0282	0.6548 ± 0.0197	0.3586 ± 0.0238	0.4161 ± 0.0308
GraphSage	Perf. Gap	56.1335 ± 1.0929	45.6566 ± 0.7884	49.2196 ± 2.5772	61.4682 ± 3.7748	46.5315 ± 2.6960	54.7559 ± 4.0392
	Adv.	0.8206 ± 0.0127	0.5916 ± 0.0201	0.5038 ± 0.0222	0.8713 ± 0.0214	0.5803 ± 0.0335	0.5263 ± 0.0212
GAT	Perf. Gap	68.1249 ± 2.5124	23.2490 ± 2.2998	34.6459 ± 2.7079	63.3213 ± 1.5763	18.4210 ± 5.1792	31.3578 ± 2.5447
	Adv.	0.7873 ± 0.0352	0.1495 ± 0.0301	0.2046 ± 0.0301	0.7247 ± 0.0382	0.2203 ± 0.0417	0.2174 ± 0.0200

Table 24: Performance gap (in %) and membership advantage on Chameleon when 50% of nodes are in the training set of the target model and the attacker train fraction is $r = 0.8$.

Model	Metric	Random			Snowball		
		Original split	Full graph	No edges	Original split	Full graph	No edges
GCN	Perf. Gap	46.3581 ± 2.8051	20.2978 ± 2.4804	46.5397 ± 2.1297	57.4159 ± 0.8470	22.0002 ± 3.6979	47.7655 ± 2.4608
	Adv.	0.5488 ± 0.0702	0.1711 ± 0.0378	0.3789 ± 0.0346	0.6623 ± 0.0199	0.3781 ± 0.0405	0.4099 ± 0.0341
GraphSage	Perf. Gap	56.1335 ± 1.0929	45.6566 ± 0.7884	49.2196 ± 2.5772	61.4682 ± 3.7748	46.5315 ± 2.6960	54.7559 ± 4.0392
	Adv.	0.8243 ± 0.0225	0.5971 ± 0.0226	0.5111 ± 0.0338	0.8801 ± 0.0260	0.5915 ± 0.0251	0.5377 ± 0.0217
GAT	Perf. Gap	68.1249 ± 2.5124	23.2490 ± 2.2998	34.6459 ± 2.7079	63.3213 ± 1.5763	18.4210 ± 5.1792	31.3578 ± 2.5447
	Adv.	0.8044 ± 0.0388	0.1637 ± 0.0402	0.2164 ± 0.0341	0.7310 ± 0.0300	0.2515 ± 0.0526	0.2132 ± 0.0267

Table 25: Performance gap (in %) and membership advantage on Pubmed when 10% of nodes are in the training set of the target model and the attacker train fraction is $r = 0.1$.

Model	Metric	Random			Snowball		
		Original split	Full graph	No edges	Original split	Full graph	No edges
GCN	Perf. Gap	10.1342 ± 0.4471	3.3656 ± 0.5857	13.0300 ± 0.3600	8.0153 ± 0.3137	5.9776 ± 0.7850	9.2859 ± 0.8411
	Adv.	0.2686 ± 0.0207	0.0333 ± 0.0083	0.1344 ± 0.0081	0.2659 ± 0.0762	0.3243 ± 0.0643	0.1627 ± 0.0491
GraphSage	Perf. Gap	14.4430 ± 0.8746	10.3028 ± 0.7984	12.4606 ± 0.3359	19.1853 ± 0.8606	14.4240 ± 0.6453	12.5284 ± 2.5280
	Adv.	0.1996 ± 0.0150	0.1030 ± 0.0139	0.1310 ± 0.0135	0.3206 ± 0.0473	0.2259 ± 0.0548	0.1512 ± 0.0412
GAT	Perf. Gap	13.5540 ± 0.4784	3.6067 ± 0.6083	14.1310 ± 0.4605	11.6681 ± 0.2751	7.7704 ± 1.2851	10.2044 ± 1.0122
	Adv.	0.1615 ± 0.0181	0.0301 ± 0.0098	0.1502 ± 0.0073	0.2267 ± 0.0658	0.1912 ± 0.0656	0.1527 ± 0.0529

Table 26: Performance gap (in %) and membership advantage on Pubmed when 10% of nodes are in the training set of the target model and the attacker train fraction is $r = 0.2$.

Model	Metric	Random			Snowball		
		Original split	Full graph	No edges	Original split	Full graph	No edges
GCN	Perf. Gap	10.1342 ± 0.4471	3.3656 ± 0.5857	13.0300 ± 0.3600	8.0153 ± 0.3137	5.9776 ± 0.7850	9.2859 ± 0.8411
	Adv.	0.3215 ± 0.0230	0.0311 ± 0.0072	0.1394 ± 0.0090	0.2774 ± 0.0727	0.3384 ± 0.0612	0.1732 ± 0.0515
GraphSage	Perf. Gap	14.4430 ± 0.8746	10.3028 ± 0.7984	12.4606 ± 0.3359	19.1853 ± 0.8606	14.4240 ± 0.6453	12.5284 ± 2.5280
	Adv.	0.2099 ± 0.0177	0.1004 ± 0.0112	0.1368 ± 0.0122	0.3255 ± 0.0488	0.2348 ± 0.0524	0.1623 ± 0.0333
GAT	Perf. Gap	13.5540 ± 0.4784	3.6067 ± 0.6083	14.1310 ± 0.4605	11.6681 ± 0.2751	7.7704 ± 1.2851	10.2044 ± 1.0122
	Adv.	0.1825 ± 0.0159	0.0313 ± 0.0076	0.1523 ± 0.0077	0.2377 ± 0.0621	0.2107 ± 0.0551	0.1606 ± 0.0540

Table 27: Performance gap (in %) and membership advantage on Pubmed when 10% of nodes are in the training set of the target model and the attacker train fraction is $r = 0.5$.

Model	Metric	Random			Snowball		
		Original split	Full graph	No edges	Original split	Full graph	No edges
GCN	Perf. Gap	10.1342 ± 0.4471	3.3656 ± 0.5857	13.0300 ± 0.3600	8.0153 ± 0.3137	5.9776 ± 0.7850	9.2859 ± 0.8411
	Adv.	0.3752 ± 0.0181	0.0457 ± 0.0094	0.1444 ± 0.0093	0.2864 ± 0.0685	0.3503 ± 0.0622	0.1819 ± 0.0530
GraphSage	Perf. Gap	14.4430 ± 0.8746	10.3028 ± 0.7984	12.4606 ± 0.3359	19.1853 ± 0.8606	14.4240 ± 0.6453	12.5284 ± 2.5280
	Adv.	0.2326 ± 0.0250	0.1141 ± 0.0100	0.1433 ± 0.0099	0.3252 ± 0.0494	0.2370 ± 0.0537	0.1731 ± 0.0319
GAT	Perf. Gap	13.5540 ± 0.4784	3.6067 ± 0.6083	14.1310 ± 0.4605	11.6681 ± 0.2751	7.7704 ± 1.2851	10.2044 ± 1.0122
	Adv.	0.2107 ± 0.0193	0.0381 ± 0.0083	0.1620 ± 0.0082	0.2487 ± 0.0634	0.2237 ± 0.0548	0.1672 ± 0.0528

Table 28: Performance gap (in %) and membership advantage on Pubmed when 10% of nodes are in the training set of the target model and the attacker train fraction is $r = 0.8$.

Model	Metric	Random			Snowball		
		Original split	Full graph	No edges	Original split	Full graph	No edges
GCN	Perf. Gap	10.1342 ± 0.4471	3.3656 ± 0.5857	13.0300 ± 0.3600	8.0153 ± 0.3137	5.9776 ± 0.7850	9.2859 ± 0.8411
	Adv.	0.3930 ± 0.0303	0.0581 ± 0.0146	0.1618 ± 0.0156	0.3051 ± 0.0716	0.3659 ± 0.0633	0.1841 ± 0.0631
GraphSage	Perf. Gap	14.4430 ± 0.8746	10.3028 ± 0.7984	12.4606 ± 0.3359	19.1853 ± 0.8606	14.4240 ± 0.6453	12.5284 ± 2.5280
	Adv.	0.2627 ± 0.0189	0.1251 ± 0.0189	0.1545 ± 0.0126	0.3281 ± 0.0567	0.2442 ± 0.0578	0.1818 ± 0.0339
GAT	Perf. Gap	13.5540 ± 0.4784	3.6067 ± 0.6083	14.1310 ± 0.4605	11.6681 ± 0.2751	7.7704 ± 1.2851	10.2044 ± 1.0122
	Adv.	0.2286 ± 0.0223	0.0532 ± 0.0164	0.1785 ± 0.0186	0.2550 ± 0.0677	0.2378 ± 0.0571	0.1749 ± 0.0613

Table 29: Performance gap (in %) and membership advantage on Pubmed when 50% of nodes are in the training set of the target model and the attacker train fraction is $r = 0.1$.

Model	Metric	Random			Snowball		
		Original split	Full graph	No edges	Original split	Full graph	No edges
GCN	Perf. Gap	2.5683 ± 0.2999	0.8667 ± 0.4915	2.9532 ± 0.3993	3.9046 ± 0.4097	1.9434 ± 0.6703	1.9327 ± 0.2369
	Adv.	0.0272 ± 0.0080	0.0079 ± 0.0037	0.0222 ± 0.0064	0.1626 ± 0.0159	0.2632 ± 0.0085	0.0446 ± 0.0123
GraphSage	Perf. Gap	5.6489 ± 0.3526	3.7619 ± 0.4879	3.3778 ± 0.3162	10.4561 ± 0.6708	5.8930 ± 0.2408	3.6835 ± 0.8331
	Adv.	0.0508 ± 0.0085	0.0305 ± 0.0075	0.0250 ± 0.0059	0.4010 ± 0.0143	0.0738 ± 0.0108	0.0411 ± 0.0082
GAT	Perf. Gap	3.0837 ± 0.3448	0.8073 ± 0.4878	3.1031 ± 0.2361	4.9210 ± 0.5259	3.1546 ± 0.5185	2.1683 ± 0.3813
	Adv.	0.0310 ± 0.0079	0.0100 ± 0.0046	0.0245 ± 0.0067	0.1718 ± 0.0158	0.0651 ± 0.0138	0.0448 ± 0.0116

Table 30: Performance gap (in %) and membership advantage on Pubmed when 50% of nodes are in the training set of the target model and the attacker train fraction is $r = 0.2$.

Model	Metric	Random			Snowball		
		Original split	Full graph	No edges	Original split	Full graph	No edges
GCN	Perf. Gap	2.5683 ± 0.2999	0.8667 ± 0.4915	2.9532 ± 0.3993	3.9046 ± 0.4097	1.9434 ± 0.6703	1.9327 ± 0.2369
	Adv.	0.0298 ± 0.0059	0.0070 ± 0.0045	0.0234 ± 0.0045	0.1745 ± 0.0128	0.2942 ± 0.0168	0.0530 ± 0.0091
GraphSage	Perf. Gap	5.6489 ± 0.3526	3.7619 ± 0.4879	3.3778 ± 0.3162	10.4561 ± 0.6708	5.8930 ± 0.2408	3.6835 ± 0.8331
	Adv.	0.0538 ± 0.0031	0.0338 ± 0.0059	0.0268 ± 0.0038	0.4180 ± 0.0140	0.0806 ± 0.0135	0.0474 ± 0.0074
GAT	Perf. Gap	3.0837 ± 0.3448	0.8073 ± 0.4878	3.1031 ± 0.2361	4.9210 ± 0.5259	3.1546 ± 0.5185	2.1683 ± 0.3813
	Adv.	0.0381 ± 0.0093	0.0106 ± 0.0058	0.0256 ± 0.0048	0.1904 ± 0.0120	0.0759 ± 0.0132	0.0515 ± 0.0107

Table 31: Performance gap (in %) and membership advantage on Pubmed when 50% of nodes are in the training set of the target model and the attacker train fraction is $r = 0.5$.

Model	Metric	Random			Snowball		
		Original split	Full graph	No edges	Original split	Full graph	No edges
GCN	Perf. Gap	2.5683 ± 0.2999	0.8667 ± 0.4915	2.9532 ± 0.3993	3.9046 ± 0.4097	1.9434 ± 0.6703	1.9327 ± 0.2369
	Adv.	0.0386 ± 0.0057	0.0099 ± 0.0045	0.0293 ± 0.0061	0.1940 ± 0.0110	0.3209 ± 0.0105	0.0593 ± 0.0101
GraphSage	Perf. Gap	5.6489 ± 0.3526	3.7619 ± 0.4879	3.3778 ± 0.3162	10.4561 ± 0.6708	5.8930 ± 0.2408	3.6835 ± 0.8331
	Adv.	0.0604 ± 0.0037	0.0402 ± 0.0077	0.0333 ± 0.0058	0.4399 ± 0.0157	0.0879 ± 0.0140	0.0495 ± 0.0087
GAT	Perf. Gap	3.0837 ± 0.3448	0.8073 ± 0.4878	3.1031 ± 0.2361	4.9210 ± 0.5259	3.1546 ± 0.5185	2.1683 ± 0.3813
	Adv.	0.0491 ± 0.0095	0.0144 ± 0.0057	0.0330 ± 0.0061	0.2070 ± 0.0091	0.0830 ± 0.0119	0.0591 ± 0.0102

Table 32: Performance gap (in %) and membership advantage on Pubmed when 50% of nodes are in the training set of the target model and the attacker train fraction is $r = 0.8$.

Model	Metric	Random			Snowball		
		Original split	Full graph	No edges	Original split	Full graph	No edges
GCN	Perf. Gap	2.5683 ± 0.2999	0.8667 ± 0.4915	2.9532 ± 0.3993	3.9046 ± 0.4097	1.9434 ± 0.6703	1.9327 ± 0.2369
	Adv.	0.0411 ± 0.0123	0.0203 ± 0.0116	0.0354 ± 0.0114	0.2023 ± 0.0115	0.3261 ± 0.0121	0.0667 ± 0.0094
GraphSage	Perf. Gap	5.6489 ± 0.3526	3.7619 ± 0.4879	3.3778 ± 0.3162	10.4561 ± 0.6708	5.8930 ± 0.2408	3.6835 ± 0.8331
	Adv.	0.0641 ± 0.0109	0.0449 ± 0.0105	0.0373 ± 0.0089	0.4482 ± 0.0163	0.0916 ± 0.0167	0.0629 ± 0.0108
GAT	Perf. Gap	3.0837 ± 0.3448	0.8073 ± 0.4878	3.1031 ± 0.2361	4.9210 ± 0.5259	3.1546 ± 0.5185	2.1683 ± 0.3813
	Adv.	0.0538 ± 0.0160	0.0240 ± 0.0126	0.0387 ± 0.0106	0.2215 ± 0.0104	0.0883 ± 0.0139	0.0672 ± 0.0132