# How Feasible is Augmenting Fake Nodes with Learnable Features as a Counter-strategy against Link Stealing Attacks?

Mir Imtiaz Mostafiz mmostafi@purdue.edu Department of Computer Science Purdue University West Lafayette, Indiana, USA Imtiaz Karim karim7@purdue.edu Department of Computer Science Purdue University West Lafayette, Indiana, USA Elisa Bertino bertino@purdue.edu Department of Computer Science Purdue University West Lafayette, Indiana, USA

# Abstract

Graph Neural Networks (GNNs) are widely used and deployed for graph-based prediction tasks. However, as good as GNNs are for learning graph data, they also come with the risk of privacy leakage. For instance, an attacker can run carefully crafted queries on the GNNs and, from the responses, can infer the existence of an edge between a pair of nodes. This attack, dubbed as a link-stealing attack, can jeopardize the user's privacy by leaking potentially sensitive information. To protect against this attack, we propose an approach called Node Augmentation for Restricting Graphs from Insinuating their Structure (NARGIS) and study its feasibility. NARGIS is focused on reshaping the graph embedding space so that the posterior from the GNN model will still provide utility for the prediction task but will introduce ambiguity for the link-stealing attackers. To this end, NARGIS applies spectral clustering on the given graph to facilitate it being augmented with new nodes- that have learned features instead of fixed ones. It utilizes tri-level optimization for learning parameters for the GNN model, surrogate attacker model, and our defense model (i.e. learnable node features). We extensively evaluate NARGIS on three benchmark citation datasets over eight knowledge availability settings for the attackers. We also evaluate the model fidelity and defense performance on influence-based link inference attacks. Through our studies, we have figured out the best feature of NARGIS- its superior fidelity-privacy performance tradeoff in a significant number of cases. We also have discovered in which cases the model needs to be improved, and proposed ways to integrate different schemes to make the model more robust against link stealing attacks.

### 1 Introduction

Graph-based data representations are widely used in online products [10], social networks [14], content services [23], and web services [12]. For learning graph-based data, different kinds of Graph Neural Network (GNN) architectures, e.g., GCN [7], GAT [16], SAGE [4] have been deployed in these services to predict user preferences, suggest preferable products, and enhance community structure. These online services gather user data that often contain sensitive information and are often of interest to malicious parties. Hence, these services can sometimes be provided through an API [11] instead of direct access to the models for security and privacy purposes. Nonetheless, both API and model access are still vulnerable to leaking sensitive information through the GNNs [5, 22]. When a GNN is trained on a particular graph dataset, malicious parties can run carefully crafted queries on that GNN's API and, from the responses, can reverse-engineer the graph structures. These forms of attacks are known as Graph Reconstruction Attacks (GIA) [5, 28, 29],

where an adversary has some prior (i.e., node features, dataset name, etc.) or posterior (i.e., APIs providing the probability of what movie a user is likely to watch in a streaming platform etc.) knowledge to recover the relations in the graph. One form of graph reconstruction attack is the link-stealing attack (also known as edge inference attack), introduced by He et al. [5], where an attacker can run queries on a GNN trained on a graph dataset for node classification tasks. recover the prediction probabilities per node (known and referred to as posteriors), and run similarity measure-based unsupervised and supervised learning algorithms to predict the existence of an edge between a pair of nodes. Link-stealing attacks have quantitatively shown the vulnerabilities of graph structures' privacy while used as a dataset for GNN learning. Attackers even often use surrogate models trained on another graph from the same domain to replicate the original GNN functionalities and infer the training graph structure.

Prior Defenses. To protect against link-stealing attacks that leverage node posteriors, several approaches have been proposed based on Differential Privacy (DP). Zhu et al. [30] introduced link-local differential privacy, where one introduces noise in the localized graph's adjacency matrix of decentralized nodes, for training a GNN in a central server without revealing the exact existence of edges. Kolluri et al. [9] proposed a new Multi-layer perceptron (MLP) based architecture, where the edge information is condensed in clustering-ingrained features and fed into MLPs for node classification task, enabling learning on graphs while obfuscating edge information. While the approach by Zhu et al. [30] achieves defense guarantees in localized settings, a requirement of this defense is that the nodes perturb their adjacency lists before sending them to the server. Also, in the approach by Kolluri et al. [9], edge information is not explicitly transmitted across the network- rather, they are compressed into a latent space representation before being transmitted. In both cases, the original edge information is modified or lost, and thus the expressiveness of the learned representation of the concerned graphs is reduced. Wu et al. [22] discussed different Differentially Private Graph Convolutional Neural Networks (DP-GCN) mechanisms (EdgeRand, LapGraph) for defending against edge inference attacks. While their approach achieves DP guarantee, both of them have to trade off model utility highly ( $\epsilon \leq 1$ ) to safeguard privacy, or sacrifice privacy to attain higher utility ( $\epsilon >= 6$ ). Also, *EdgeRand* changes the graph density extensively, often creating out-of-memory error for large graphs.

**Defense Key Insight.** A key insight in the DP-based defenses is the injection of noises to defend the graph structure from being inferred. These noises are added in the adjacency matrix of the graphs [22], which can change the sparsity of the graph to a huge extent (*EdgeRand*) or delete some old edges (*LapGraph*). It is to be noted that noise injection can also be a form of attack. Noisebased perturbation as an attack form is referred to as the *poisoning attacks*. Poisoning attacks modify the graph structures at training time by changing the node features or even inserting fake nodes, leading to compromising the learning ability of the GNNs. The key takeaway is that a defense (noise addition) can also be a form of attack. As DP-based approaches' noise addition has been performed on the edge perspective, it now raises a counter-question, *Can a node-based noise addition (i.e., a poisoning attack through changing node features or adding fake nodes) also be a form of defense against link stealing attacks for the Graphs*?

To answer this question, we analyzed a form of graph modification attacks known as Graph Injection Attacks (GIA) [15, 18, 19, 31], where the adversary inserts carefully crafted nodes in a graph (i.e., publishing a fake paper to perturb a citation network, etc.) to mislead the prediction of GNNs through the perturbation of posteriors. The intuition is that if a defense mechanism can take the role of attacker and insert fake nodes into the graph on which the GNN will be trained, it can "counter-attack" the attacker. However, it will also lead to the GNN prediction model being sub-optimized for the original task, as learning the graph with perturbed topology will cause the node posteriors to be perturbed, too. So, the problem can posed as a bi-optimization: perturbing the graph structure and GNN posteriors enough to defend against the attackers but also to provide optimal service to the user within a range. Defenses for ML models through bi-level optimization have been proposed by Wu et al. [21]. Interestingly, it works by learning a noise transition matrix for posterior perturbations. However, this defense is tailored against model-stealing attacks for images and does not cover the graph domain. Also, the optimization is done after the learning to hide the model from being inferred, whereas our target is to optimize during training to hide the training data structure. Moreover, the user task and the attacker task are the same (image classification) in the stated model. In contrast, in our model, the user focus is to learn to predict node classes, and the attacker's focus is on edge inference.

Motivating Example for Our Approach For example, in Figure 1, three nodes  $u_a, u_b, u_c \in V$  have the posterior output vectors for a two-class classification problem as  $\mathbf{a} = [0.3, 0.7], \mathbf{b} = [0.1, 0.9], \&$  $\mathbf{c} = [0.4, 0.6]$ , respectively, and only node pair  $(u_a, u_c)$  have an edge between them in their corresponding graph (none of the nodes are drawn in the figure as per scale and orientation). The Euclidean distance between the posteriors are (pairwise):  $(d_{ab}, d_{bc}, d_{ca}) =$ (0.28, 0.42, 0.14). A link-stealing attacker can set a distance upperbound threshold of 0.2, and hypothesize correctly that the node pair  $(u_a, u_c)$  have an edge between them (as only these nodes are not more distant than the threshold). But if the embeddings are perturbed in a manner such that the posteriors become  $\mathbf{a}' = [0.13, 0.87]$ ,  $\mathbf{b}' = [0.35, 0.65] \& \mathbf{c}' = [0.45, 0.55]$ , then the classes will remain same still (the second entry being always the highest, inferring all have the label 1 from between  $\{0, 1\}$ ), but the Euclidean distance would become  $(d_{a'b'}, d_{b'c'}, d_{c'a'}) = (0.31, 0.14, 0.45)$  Hence if the link stealer has a threshold of 0.2 again, it will likely conclude that there is an edge between node pair  $(u_b, u_c)$  and the attack is thus thwarted.



Figure 1: Illustration of Posterior Perturbation in Posterior Simplex Space for Protection against Link Stealing Attacks

Our Approach. Our approach focuses on reshaping the graph embedding space in such a way that the posteriors from the GNN model will still provide utility for node prediction but will introduce ambiguity for the link-stealing attackers. As the graph embedding space is constructed through message passing among the nodes, and in a k – *layer* GNN, this information exchange ranges to the whole k-hop neighborhood of a graph; our intuition is that we can augment new nodes with carefully crafted features, whose introduction can perturb the embedding space of a cluster of nodes in their own k-neighborhoods. we integrate a form of poison attack in the design of our defense, as our research question demanded. The idea of inserting nodes with crafted features resembles the node injection attack introduced by Zou et al. [31], where the features are learned through optimization instead of being fixed. Besides, to ensure that our new nodes can influence a significant neighborhood in a graph, we apply spectral clustering on the graph first and augment edges from the new node to the center node of the formed clusters. This strategy is different from the one applied by Zou et al. [31], where topologically defected edges are selected under different criteria. Moreover, we introduce a tri-level optimization for learning parameters for our defense model (learnable node features), the model parameters for the target GNN task, and also for a surrogate attacker model which helps to simulate the possible attacks in order to be prepared against. We name our approach as "Node Augmentation for Restricting Graphs from Insinuating their Structure" (NARGIS). We evaluate NARGIS on three benchmark citation dataset: Cora, Citeseer, Pubmed [25]. We also evaluated our approach on LinkTeller [22], a state-of-the-art link inference attack. Through our evaluations, we have found some cases where our model and their variants can better the performance of Differential-Privacy (DP) based defenses. Also, we could find out the limitations of our model while defending against some form of attacks.

**Contributions.** This paper makes the following technical contributions.

 We studied how a form of Graph modification attack can be integrated as a defense for a Graph link-stealing or edge inference attack. To this end, we have introduced NARGIS-a node augmentation-based defense for Graph Neural Networks, which can restrict the model from providing posteriors leading to edge inference by attackers, by perturbing the embedding space through *augmenting* nodes and *learning* their features.

- We have evaluated our approach on three citation datasets: Cora, Citeseer and Pubmed [25], on eight attacking settings stated in [5] and on *LinkTeller* [22]. We also evaluated node prediction performances on the augmented model.
- We propose that NARGIS can be tuned to get near-optimal defensive performances as the DP-based Defenses, with greater fidelity.
- We also discussed the cases where the model needs to improve and how different schemes can be used in further investigation to make it perform better.

### 2 Preliminaries

In this section, we introduce preliminary definitions of Graph Learning, and present the scenarios when the privacy of user data, encoded in the graph concerned, are compromised. We then define the attack our model is trying to defend from.

### 2.1 Graph Learning

2.1.1 *Graph.* A graph *G* is an ordered tuple (V, E) where *V* is the vertex set and *E* is the edge set. Edges can be represented by *Adjacency Matrix* A; for a graph with *n* nodes, i.e., |V| = n,  $A \in \{0, 1\}^{n \times n}$ . Formally, for A,  $A_{ij} = [(v_i, v_j) \in E]$ . Node attributes are represented as a matrix  $X \in \mathbb{R}^{n \times d}$  with feature dimension  $d \in \mathbb{N}$ . The label set  $Y \in \{0, 1, \dots, c-1\}^{n \times 1}$  is a column vector denoting the label of each node (from candidate labels  $C = \{0, 1, \dots, c-1\}$ ). Thus we can represent a Graph *G* as an ordered 6-tuple (V, A, X, Y, n, c).

2.1.2 Node Embedding. An embedding function is a mapping from a high dimensional space to a low-dimensional one that can preserve particular properties of the domain space. For a graph G, a Node Embedding is a function  $f : G \to \mathcal{H} \in \mathbb{R}^{n \times d_e}$ , where  $d_e << d$  is the embedding dimension and  $\mathcal{H}$  is the Embedding space. For a graph, the most known practice is to find an embedding that preserves the neighborhood similarities or distances among the nodes.

2.1.3 Graph Neural Network (GNN) Learning. A Graph Neural Network (GNN) is a neural network, which takes a graph G as input and generates a node embedding  $f : G \to \mathcal{H}$  for each of its nodes, from the edge relationships. The most popular mechanism for learning these embeddings is called *Message-Passing*. Under this mechanism, in each layer of the **GNN**, for every single node, two operations are performed: (1) **Aggregation**: the concerned node's neighbors' node embeddings from the previous layer are collected and combined; (2) **Update**: the combined neighborhood node embeddings, along with the concerned node's previous layer embedding, are used to calculate the node's embedding for this current layer. In this way, nodes related to each other through multiple hops of the graph can influence each other in the latent embedding space. Formally, for a *L*-layered **GNN** ( $L \in \mathbb{N}$ ), the calculation of the hidden representation of a node  $v \in V$  in the  $l \in \{1, 2, ..., L - 1\}$ , denoted as  $h_v^{(l)}$ , (where  $N_v$  means the neighborhood of v in G, i.e.  $N_v = \{u \in V | (v, u) \in E\}$ ),

is executed as follows:

$$v_{\mathcal{N}}^{(l-1)} = \operatorname{aggregate}(\underset{\substack{u \in \\ \mathcal{N}_{v}}}{\forall} h_{u}^{(l-1)}), h_{v}^{(l)} = \operatorname{update}(h_{v}^{(l-1)}, v_{\mathcal{N}}^{(l-1)}) \quad (1)$$

Where UPDATE, AGGREGATE are designated differentiable functions. As an initial value,  $h_v^{(0)} = \mathbf{X}_v$  is chosen. Notable *Message-passing* **GNNs** include Graph Convolutional Network (GCN) [7], Graph Attention Network (GAT) [16], Graph Isomorphism Network (GIN) [24], Graph Sample and Aggregate (GraphSAGE) [4].

In this work, we focus on two of the most important graph learning tasks: (1) *Node Classification*: a node's attached class label is predicted by training on a small subset of labeled nodes, and in the process, node embeddings are learned; and, (2) *Link Prediction*: we learn the embeddings of a node, and for a pair of nodes, we train a neural network or any unsupervised method to find out whether an edge exists between them. The defense side's model is based on node classification, whereas the attacker focuses on the link prediction task.

# 2.2 Attacker Model

We now describe the threat model of our work in terms of the attack model's goal, knowledge, and capabilities. The model in consideration is a GNN, representing a given graph (e.g., a social network or online shopping recommender), trained for the node classification task. At the last layer of the GNN, for each node in the graph, we get a probability distribution (posterior) of the candidate set of node classes. As an example, for a citation network, the nodes representing the works maybe classified as either one from the set *[Representation Learning, Reinforcement Learning, Meta-Learning]*, and hence the posterior probability distribution will be a three-dimensional vector representing a simplex.

2.2.1 Attacker Goal. The attacker will try to infer the graph structure, i.e., finding the existence of an edge between two given nodes. If the attacker has knowledge about the nodes and for each pair of nodes can infer whether there exists an edge between them, then they can eventually reconstruct the whole graph. For social or product recommendation networks, these edges can be of high interest for malicious parties. The attack goal is neither under poison settings (e.g., corrupting the graph), nor under evasion settings (e.g., evading the defense). Rather, it is a reconstruction settingaiming to find out the graph's structure.

2.2.2 Attacker Knowledge. The attacker's knowledge can be discussed along two different aspects: the GNN model and the training data/Graph.

The attacker is completely in the dark about the GNN model parameter and hyperparameters. The attacker can access to the posterior distribution of node classes from the GNN model.

In case of the data or the graph needed for training, the attacker can face multiple scenarios as stated in [5], depending on the availability of notable Graph constituents. Namely, they are: (1) **Target Dataset's Nodes' Attributes**  $\mathcal{F}$ : The attacker sometimes can have the knowledge about the attributes of the nodes  $\mathcal{F}$  and labels of a small subset of data used to train the GNN model. (2) **target Dataset's Partial Graph**  $\mathcal{A}$ : A subset of graph edges can also be provided to the attacker, to be used as ground truth edges to train the link inference model and (3) **Shadow Dataset**  $\mathcal{D}'$ : a shadow dataset can be provided to facilitate transfer learning to mimic the original graph. The attacker trains a shadow target model from a graph with own nodes and edges, from same or different domain. Depending on the presence/absence of these three auxiliaries,  $2^3 = 8$  attacking scenarios has been considered in [5]. The background knowledge is represented as an ordered tuple  $\mathcal{K} = (\mathcal{F}, \mathcal{A}, \mathcal{D}')$ , which can have eight different values ranging from  $(\times, \times, \times)$  (i.e., all absent) to  $(\mathcal{F}, \mathcal{A}, \mathcal{D}')$  (i.e., all present). All the possible settings are described in Table 1. Also, in our experimental study (Section 5), we have described all the attack scenarios before discussing our model's performance under these settings.

### Table 1: Attacker Knowledge Configurations for Link Stealing Attacks, as described in [5]

Attack No.	Node Attribute, $\mathcal{F}$	Partial Graph, Â	Shadow Dataset, $\mathcal{D}'$	Background Knowledge, K	Attack No.	Node Attribute, $\mathcal{F}$	Partial Graph, <i>Â</i> l	Shadow Dataset, D'	Background Knowledge, $\mathcal{K}$
Attack-0	×	×	×	$(X \times X)$	Attack-4	×	./	./	(× A D')
Attack-1	×	x	Ŷ	$(X, X, \mathcal{D}')$	Attack-5	$\widehat{\checkmark}$	×	٠ ٧	$(\mathcal{F}, X, \mathcal{D}')$
Attack-2	$\checkmark$	×	×	$(\hat{\mathcal{F}}, \mathbf{x}, \mathbf{x})$	Attack-6	$\checkmark$	$\checkmark$	×	$(\mathcal{F}, \mathcal{A}, \times)$
Attack-3	×	$\checkmark$	×	$(\times, \mathcal{A}, \times)$	Attack-7	$\checkmark$	$\checkmark$	$\checkmark$	$(\mathcal{F},\mathcal{A},\mathcal{D}')$

2.2.3 Attacker Capabilities. The attacker can only run queries on the GNN model to find out the posteriors of the predicted node classes. Augmented with the background knowledge, as described in Section 2.2.2, the attacker will try to infer the graph edges.

2.2.4 Attack Type. The attacker will try a similarity-based attack by leveraging the heuristic that nodes with similar posterior class distribution are expected to have edges between them. This type of attack is called **link stealing attack** as per the literature [5] and [26]. The attacker uses different similarity or distance metrics(e.g., cosine similarity, correlation coefficient) to find similar node-pairs. Formally, given a black box **GNN** model  $\mathcal{G}$ , the training (target) graph nodes V, the nodes' posteriors from  $\mathcal{G}$ , the background knowledge  $\mathcal{K}$ , and two nodes  $u, v \in V$ , the link stealing attack allows one to determine whether there exists an edge in the target graph between u and v.

### 3 Challenges and Solutions

The design of our approach, based on the idea of adding new, fake nodes in the graph with crafted features to perturb the embedding space enough to thwart the link-stealing attacker but be sufficiently accurate enough for the model user, requires addressing several challenges: (C1) Where should we augment the nodes? The k-layer embeddings of a node depend on its k-hop neighbors. Therefore, any change in the embeddings of its neighbor nodes also changes the embedding of the node. Thus, we need to connect new nodes to such nodes in the original graph that are already connected with a significant number of nodes through a minimal number of hops. To address this challenge, we apply spectral clustering [17] to partition the graph into different clusters and find the cluster centers (the node with the least intra-cluster average distance). Then, we add edges from the new nodes to the cluster center. As a cluster center itself is connected to many nodes with one or two hops, connecting an edge from a new node to this node will help to perturb the embeddings of the nodes in its proximity efficiently. (C2) What should be an optimal bound for the number of new nodes? We need

to find the optimal number of new nodes so that each of them, influencing their cluster, can collectively perturb the whole graph's embedding and posterior space. We hypothesize that this optimal number depends on the *graph's density*, *which is roughly inversely proportional* to it. As in a dense graph, more nodes can be connected in one or two hops in message-passing scheme for GNNs due to higher number of edges, less cluster-center nodes are needed to perturb the embedding and posterior spaces of neighborhood nodes. We state and prove the theoretical result below:

PROPOSITION 1. Let the Spectral Clustering Algorithm [17] be applied on an unweighted graph G = (V, E) in such a way that (i) every cluster is equally (approximately) sized in terms of the number of nodes, (ii) each node in a cluster is within the L-neighborhood of other nodes in the same cluster for a fixed  $L \in \mathbb{N}$ , and (iii) the highest degree possible within a cluster for a node is K for a fixed  $K \in \mathbb{N}$ , then to ensure the mentioned constraints, the number of clusters needed to form, c, is inversely proportional to the graph density  $\sigma$ , i.e.

$$c \propto \frac{1}{\sigma}$$

**PROOF.** The number of nodes and edges in *G* are n = |V|, e = |E|, respectively. Let the lowest and highest possible number of nodes in a cluster be  $n_c^{min} \& n_c^{max}$ , respectively. Then *n* can be bounded as,

$$n_c^{min} * c \le n \le n_c^{max} * c \tag{2}$$

Let the expected number of edges within a cluster to form a minimally connected component following the *L*-neighborhood and *K*-maximum degree assumption is  $e_c$ . If each cluster is considered as a super-node, and these super-nodes are connected to form minimum-spanning tree of clusters- then for a graph, there are three kinds of edges formed: intra-cluster minimally constrained connected component edges, inter-cluster minimum spanning tree edges, and the extra inter-cluster edges (not necessary to form the tree of clusters). As there are *c* clusters, there will be c - 1 intercluster minimum-spanning tree edges. We can bound the number of edges *e* as,

$$c * e_c + (c - 1) \le e \tag{3}$$

To deduce the lower and upper bound for  $e_c$ , we consider two cases of the edge structure of the clusters. As a lower bound case, we consider the minimum spanning tree of the nodes in a cluster; and as an upper bound case, we consider the complete K - ary tree of the nodes. For the second case, let the highest degree (intra-cluster) node be the root. Then there will be at most K nodes connected with it in the first level. In the next level, every node will have K - 1children (as the highest intra-cluster degree is K and they already have a parent), so there will be K \* (K - 1) edges. As all the nodes are within an *L*-neighborhood, the tree will have *L* levels, and the total number of edges will be  $K + K * (K - 1) + K * (K - 1)^2 + \ldots +$  $K * (K-1)^{(L-1)} = \frac{K * \{(K-1)^{L}-1\}}{K-2} = f_{max}(K,L)$ . For the first case, we cannot get an explicit formula for the number of edges in terms of K, L as it depends on the graph structure. Nevertheless, we will denote it as  $f_{min}(K, L, G)$ . As both case denotes a tree, the number of nodes will be 1 more than the number of edges- so  $n_c^{max} = 1 + e_c^{max} = f_{max}(K, L), n_c^{min} = 1 + e_c^{min} = 1 + f_{min}(K, L, G)$ , and The density  $\sigma$  of a Graph is,

How Feasible is Augmenting Fake Nodes with Learnable Features as a Counter-strategy against Link Stealing Attacks?

$$\sigma = \frac{e}{n*(n-1)} \approx \frac{e}{n^2}.$$
(4)

If every cluster is formed as the first case (minimum spanning tree), then from Equations (2) to (4), the lowest bounded density will be  $\sigma_{min} \approx \frac{e^{min}}{(n^{min})^2} = \frac{c*e_c^{min}+(c-1)}{(n_c^{min}*c)^2} = \frac{c*(e_c^{min}+1)-1}{(n_c^{min}*c)^2} = \frac{c*n_c^{min}-1}{(n_c^{min}*c)^2} \approx \frac{c*n_c^{min}}{(n_c^{min}*c)^2} = \frac{1}{n_c^{min}*c} = \frac{1}{c*f_{min}(K,L,G)}$ , and the highest bound will be,  $\sigma_{max} \approx \frac{e^{max}}{(n^{max})^2} = \frac{c*e_c^{max}+(c-1)}{(n_c^{max}*c)^2} = \frac{c*(e_c^{max}+1)-1}{(n_c^{max}*c)^2} = \frac{c*n_c^{max}}{(n_c^{max}*c)^2} = \frac{1}{n_c^{max}*c} = \frac{1}{c*f_{max}(K,L)}$ 

As from the assumptions, the Graph is fixed and so are the values of *K*, *L*. So,  $f_{min}(K, L, G)$ ,  $f_{max}(K, L)$  are both constants. Hence in both cases,  $c \propto \frac{1}{\sigma}$ 

Therefore, if two graphs  $G_1$ ,  $G_2$  have densities  $\delta_1$ ,  $\delta_2$ , respectively, and defending  $G_1$  is achieved optimally with N new nodes, then  $G_2$  should be augmented with approximately  $\lfloor \frac{N\delta_1}{\delta^2} \rfloor$  nodes. (C3) How can the competing objectives of optimizing model utility and defending link-stealing attacks be achieved? The target GNN model's posteriors are optimized for the node prediction task. However, as good as these posteriors are, homophily (edge endpoints having the same labels) is usually found in graph datasets, which exposes their vulnerability in similarity-based attacks. Thus, the posteriors individually have to be optimized but jointly have to be misleading. To address the competing objectives, we introduce a tri-level optimization of augmented node features, target GNN parameters, and surrogate attacker model parameters. We optimize augmentation (and GNN parameters), along with surrogate attacker parameters, interchangeably so that one model's gradient feedback updates the other one.

To summarize, our approach needs functionalities for graph clustering and a multi-level optimization loop to maximize the target model's utility and defensive strength.

### 4 Modulewise Detailed Operation of NARGIS

In this section, we discuss each module separately and together to form the workflow of NARGIS.

# 4.1 Augmentation Module

The Augmentation Module (see Figure 2a), takes the input graph, applies spectral clustering on it, and returns the augmented graph with new nodes and edges. In this module, the new nodes' features are still set at 0 as they have not been learned yet.

In NARGIS, we augment the graph *G* with new nodes and create some edges from the newly added nodes to the original nodes. Let  $n_{new}$  be the number of newly added nodes in the graph *G* and  $V_{new}$  be the set of newly added nodes, then  $|V_{new}| = n_{new}$ . Let the node features and labels of newly augmented nodes be, respectively,  $\mathbf{X}_{new} \in \mathbb{R}^{n_{new} \times d}$  and  $\mathbf{Y}_{new} \in \{0, 1, \dots, c-1\}^{n_{new \times 1}}$ . Also, let  $\mathbf{A}_{new} \in \{0, 1\}^{n \times n_{new}}$  be the incidence matrix between the old nodes and newly added ones, with,

$$[\mathbf{A}_{new}]_{(u,v)} = [\exists \text{ an edge}(u,v) \text{ for } u \in V \text{ and } v \in V_{new}]$$
(5)

Let  $G_{aug} = (V_{aug}, \mathbf{X}_{aug}, \mathbf{Y}_{aug}, \mathbf{A}_{aug}, n_{aug}, c)$  be the augmented graph, where

 $n_{aug} = n + n_{new}, \ V_{aug} = V \cup V_{new}, \ |V_{aug}| = n_{aug}$ (6)

$$\mathbf{X}_{aug} = \begin{bmatrix} \mathbf{X} \\ \mathbf{X}_{new} \end{bmatrix}, \ \mathbf{A}_{aug} = \begin{bmatrix} \mathbf{A} & \mathbf{A}_{new} \\ \mathbf{A}_{new}^T & \mathbf{0} \end{bmatrix}, \ \mathbf{Y}_{aug} = \begin{bmatrix} \mathbf{Y}_{new} \end{bmatrix}$$
(7)

Note that we train the **GNNs** in semi-supervised style [7], where only a portion of nodes are labeled, and among them, train, validation and test splits are made. Thus, for  $Y_{new}$ , we randomly set the labels, as they will not be part of any labeled splits. For edge augmentation, we use the Unnormalized Laplacian Based Algorithm for Spectral Clustering [17], which finds the graph clusters and their center nodes. We connect each new node with separate cluster centers to form the new edges. Function 1 represents the

ł	Function 1: $getAugmentedGraph(G)$
	Input :Graph $G = (V, \mathbf{X}, \mathbf{Y}, \mathbf{A}, n, c);$
	<b>Parameter</b> : Number of new nodes to augment, <i>n<sub>new</sub></i> ;
	Output : Augmented Graph
	$G_{aug} = (V_{aug}, \mathbf{X}_{aug}, \mathbf{Y}_{aug}, \mathbf{A}_{aug}, n_{aug}, c)$ with no
	features learned yet;
1	Cluster Center Nodes $[o_1, o_2, \dots, o_{n_{new}}] \leftarrow$ Spectral Clustering on
	G as per the algorithm in [17];
2	$V_{new} \leftarrow \{v_1, v_2, \dots, v_{n_{new}}\}$ , New nodes with zeroes set as feature;
3	Form edge set $E_{new} \leftarrow \{(o_1, v_1), (o_2, v_2), \dots, (o_{n_{new}}, v_{n_{new}})\};$
4	$X_{new} \leftarrow \{0, 0, 0, \dots, 0\}, n_{new}$ zero feature vectors;
5	$A_{new} \leftarrow \text{form adjacency matrix as per Equation (5) using}$
	$V, V_{new} \& E_{new};$
6	$Y_{new} \leftarrow n_{new}$ randomly assigned labels from 0, 1,, $c - 1$ ;
7	Form augmented graph $G_{aug} = (V_{aug}, \mathbf{X}_{aug}, \mathbf{Y}_{aug}, \mathbf{A}_{aug}, n_{aug}, c)$
	using Equations (6) and (7);
	notrong C .

8 return G<sub>aug</sub>;

functionality of the Augmentation Module.

# 4.2 Tri-Optimization based Feature Learning Module

4.2.1 Module Workflow Summary. The Tri-Optimization based Feature Learning Module is the most crucial module. Its goal is to learn the augmented nodes' features so that their existence in a GNN learning step will perturb the embedding and posterior space of the original graph's nodes. This module takes both the unlearned augmented graph (from the Augmentation Module) and main graph (training) edges and their labels (existent/non-existent) as inputs. The module consists of our target GNN model (Feature Learning Sub-Module) and a surrogate attacker model (Surrogate Attacker Sub Module)- connected through a tri-optimization-based learning loop (see Figure 2a). The detailed process is shown in Figure 2b. The target GNN model is trained for the node prediction task in the augmented graph in Stage 1. Consequently, the learned posteriors are sent to the surrogate attacker model, which combines the surrogate query edges with them to form an edge feature and then the attacker model is trained to infer the existence of the edge (Stage 2). Gradient feedback from this model is sent again to the Feature-Learning sub-module so that the augmented nodes' features are updated through gradient descent, whereas the GNN parameters are kept frozen (Stage 3). Finally, the module returns the learned features of the augmented nodes. It is to be noted that the Surrogate Attacker Sub-Module is trained only once, unlike the Feature Learning Sub-Module which is

#### Mir Imtiaz et al.



(b) Tri-Optimization based Feature Learning Model Elaborated (GNN-related inputs to Augmented Nodes' Feature, as shown in Figure 2a, are not shown here for the sake of simplicity.)

#### Figure 2: System Overview of NARGIS

trained several times. Our intuition behind this learning strategy is that the attacker's model is targeted to learn the implicit bias that similar nodes from the edges mainly- and in the first training of the Feature Learning Sub-Module, similar nodes' posteriors are trained to be similar. Hence, the surrogate attacker takes those posteriors to integrate the inductive bias regarding the homophily. But in later part of the trainings, when the Feature Learning Sub-module also gets gradient feedback to perturb the embedding and posterior space for the defense against attacks, training the surrogate attackers on those perturbed posteriors again will lead the attacker to be adaptive, which is not under our threat model settings. Making the surrogate attacker adaptive will unnecessarily weaken the defense's power, as no matter what gradient update is done in the Feature Learning Sub-module, the Surrogate Attacker Sub-module will always have an answer for that and the defensive objective will not be achieved. This procedure is elaborated in Function 2. Finally, this module returns the perturbed posteriors.

4.2.2 Module Workflow Details. In the subsequent discussions, we use  $\mathcal{D}_Q$  to denote the training edges, both positive/existent and sampled negative/non-existent edges from the main graph.

$$\mathcal{D}_{Q} = \{(u, v, y) | u \in V, v \in V, y = A[u][v]\}$$
(8)

Target GNN Model. Let  $f(G; \theta_f) : G \to [0, 1]^{n \times c}$  be the target model, which takes a graph G as input and outputs the probability distribution of the nodes' labels (i.e., posteriors). This is the model NARGIS defends. The set of learnable model parameters are denoted as  $\theta_f$  (they include both the embedding and classification layer parameters).

Augmented Target GNN Model. Let  $f_{\tau}(G_{aug}; \theta_{f_{\tau}}, \theta_{\tau}) : G_{aug} \rightarrow [0, 1]^{n_{aug} \times c}$  be the augmented target GNN model, which takes the

augmented graph  $G_{aug}$  as input and outputs the perturbed probability distribution (posteriors) of the nodes' labels. This model is used exclusively to learn the augmented node features. The set of learnable target GNN model parameters are denoted as  $\theta_{f_{\tau}}$ , and the associated learnable augmented node features are:  $\theta_{\tau}$ .

*Edge Feature Generator.* This procedure takes the node posteriors from the target GNN model and surrogate query edges to form the edge features for the *surrogate attacker model*. Formally, let  $\chi = f_{\tau}(G_{aug}; \theta_f, \theta_{\tau})$  be the node posteriors from *the augmented target model*. Then, the *Edge Feature Generator* can be denoted as a function  $k(\chi, \mathcal{D}_Q) \to \mathbb{R}^{|\mathcal{D}_Q| \times d_a}$ , where  $d_a$  be *the surrogate attack model* input dimension.

Surrogate Attack Model. We refer to  $g(u, v, k(\chi, \mathcal{D}_Q); \theta_g) \rightarrow [0, 1]$ as Surrogate Attack Model for link prediction between two nodes u, v in a (non-)existing edge in  $\mathcal{D}_Q$ .  $\theta_g$  denotes the set of learnable parameters of the model. It takes the surrogate dataset and the formed edge features as input for learning.

*Module Optimization Objective.* The goal is to learn the features of the augmented nodes. As the aim of any Learning-based model's defense is two-fold (providing utility while defending from attacks), to assist the learning of features, we learn the target model for the node classification task (to help the learnable nodes improve the utility) as well as a surrogate attacker model (to help the learnable nodes be robust against attacks).

Formally, let the node classification loss of the aforementioned Augmented Target GNN model  $f_{\tau}(.)$  be  $\mathcal{L}_{class}(G_{aug}; \theta_{f_{\tau}}, \theta_{\tau})$ . We also introduce an edge classification loss of the Surrogate Attack Model g(.) as  $\mathcal{L}_{attack}(\mathcal{D}_Q, \chi; \theta_g)$ . Lastly, based on all those learnable parameters, our loss of interest is the loss dedicated to the node augmentation-based defense,  $\mathcal{L}_{defense}(\mathcal{D}_Q, G_{aug}; \theta_g, \theta_{f_{\tau}}, \theta_{\tau})$ . The detailed definition of the loss functions are given in what follows.

Classification loss. For augmented target GNN model's classification loss  $\mathcal{L}_{class}$ , we use the negative log-likelihood loss:

$$\mathcal{L}_{class}(G_{aug};\theta_{f_{\tau}},\theta_{\tau}) \coloneqq \mathbb{E}_{u \sim V}[-log([f_{\tau}(G_{aug};\theta_{f_{\tau},\tau})]_{(u,Y_u)})] \quad (9)$$

Here, we obtain the posteriors from the *augmented target GNN model*, and for each node-label pair  $(u, Y_u) \in V_{aug} \times Y_{aug}$  of the graph  $G_{aug} = (V_{aug}, \mathbf{X}_{aug}, \mathbf{Y}_{aug}, \mathbf{A}_{aug}, n_{aug}, c)$ , we take the probability value of the true node label, and take the expected value of negative log-likelihood over all the nodes in the augmented graph that belongs actually to the original graph (as  $V \subset V_{aug}$ ).

*Surrogate Attacker Loss.* For the *surrogate attack model*'s link prediction loss, we use the binary cross-entropy loss:

$$\mathcal{L}_{attack}(\mathcal{D}_Q, \chi; \theta_g) = \mathcal{L}_{attack}(\mathcal{D}_Q, G_{aug}; \theta_g, \theta_{f_\tau}, \theta_\tau) := \mathbb{E}_{(u,v,y)\sim\mathcal{D}_Q} [\mathcal{L}_{BCE}(g(u, v, k(\chi, \mathcal{D}_Q); \theta_g), y)]$$
(10)

where  $g(u, v, k(\chi, \mathcal{D}_Q); \theta_g)$  denotes the probability of an edge between nodes  $u, v \in V$  from the surrogate model g by the attacker and y is a binary variable indicating whether there is an edge between them. Remember that as  $\chi$  is dependent on  $G_{aug}, \theta_{f_{\tau}}, \theta_{\tau}$ , this loss (and other losses stated later) depends on them too. Defender Loss. For the defender, constructing the loss is nontrivial, as we have to integrate different competing objectives. We construct four different losses for aggregating the *Defender loss*. (1) **Distribution Alignment Loss**. First, our goal is that the perturbed posterior distribution from the augmented model must be quite different from the original GNN's posterior distribution so that a similarity-metric-based attacker gets confused. So, we have to integrate a loss, which will try to maximize the difference between the posteriors  $\chi_u$ ,  $\chi_v$  for the nodes u, v, respectively, in an existent edge (u, v). The *Distribution Alignment loss for Defender* is defined as

$$\mathcal{L}_{dist}(\mathcal{D}_Q, \chi) = \mathcal{L}_{dist}(\mathcal{D}_Q, G_{aug}; \theta_{f_r}, \theta_{\tau}) := \mathbb{E}_{(u,v,y=1)\sim\mathcal{D}_Q}[-ShanonDiv(\chi_u||\chi_v)]$$
(11)

Here, *ShanonDiv*(.) is the Jensen-Shanon Divergence [20], which is a symmetric version of the Kullback-Leibler (KL) Divergence. We use Jensen-Shanon instead of KL, to make the distribution loss symmetric for the edge endpoints.

(2) Correlation Distance Loss. Second, to deviate the perturbed posterior distribution from the augmented model more from the original GNN's posterior distribution, we integrate another loss, which will try to minimize the correlation similarity between the posteriors  $\chi_u$ ,  $\chi_v$  for the nodes u, v, respectively, in an existent edge (u, v). Whereas the  $\mathcal{L}_{dist}$  is optimized to make the posterior vectors dissimilar in probability simplex, this correlation-based loss is optimized for the same task in the vector space. The *Correlation Distance loss for Defender* is defined as:

$$\mathcal{L}_{corr}(\mathcal{D}_Q, \chi) = \mathcal{L}_{corr}(\mathcal{D}_Q, G_{aug}; \theta_{f_{\tau}}, \theta_{\tau})$$
$$= \mathbb{E}_{(u,v,y=1)\sim\mathcal{D}_Q} \left[ -\left\{1 - \frac{\bar{\chi}u \cdot \bar{\chi}v}{||(\chi u - \bar{\chi}u)||_2||(\chi v - \bar{\chi}v)||_2}\right\} \right]$$
(12)

Here, we have described the formula for adjusted cosine similarity which is used interchangeably with correlation similarity. To convert it into a distance, we deduct the quantity from 1.

(3) Alignment Calibration Loss. Thirdly, while optimizing for the  $\lambda_{dist}$ , the original posterior can be perturbed in such a way that the main task can be derailed, i.e., misclassification happens. This happens if, after the perturbation, the ground truth label class's probability is not higher than the maximum one. So, we have to integrate an alignment-based loss, too, which ensures that only the class probabilities not representing the ground truth class are perturbed. As the attacker's aim is not the classification of nodes, this does not decrease much the strength of the defense. Therefore, *Alignment Calibration loss for defender* is defined as

$$\mathcal{L}_{align}(\mathcal{D}_{Q}, \mathbf{Y}, \chi) = \mathcal{L}_{align}(\mathcal{D}_{Q}, G_{aug}; \theta_{f_{\tau}}, \theta_{\tau}) := \mathbb{E}_{(u,v,y)\sim\mathcal{D}_{Q}}[-log([\chi_{u}]_{\mathbf{Y}_{u}}) - log([\chi_{v}]_{\mathbf{Y}_{v}})]$$
(13)

(4) Misclassification loss. Last, remember that merely perturbing the posterior vector is not our aim. We have to ensure that the attacker cannot predict the links' existence. So, we have to perturb in such a way that the posteriors from the model fool the surrogate model enough to misclassify the links. So, we define the *misclassification loss by the attacker for defender* as:

$$\mathcal{L}_{miss}(\mathcal{D}_Q, \chi; \theta_g) = \mathcal{L}_{miss}(\mathcal{D}_Q, G_{aug}; \theta_g, \theta_{f_t}, \theta_\tau)$$
  
$$:= \mathbb{E}_{(u,v,u) \sim \mathcal{D}_Q}[-log(1 - g(u, v, k(\chi, \mathcal{D}_Q); \theta_g)_u)]$$
(14)

Finally, taking the linear combination of the losses, we have the *Defender Loss*,

$$\mathcal{L}_{def\,ense}(\mathcal{D}_{Q}, G_{aug}; \theta_{g}, \theta_{f_{\tau}}, \theta_{\tau})$$

$$:= \begin{bmatrix} \lambda_{miss} \\ \lambda_{align} \\ \lambda_{dist} \\ \lambda_{corr} \end{bmatrix}^{T} \begin{bmatrix} \mathcal{L}_{miss}(\mathcal{D}_{Q}, G_{aug}; \theta_{g}, \theta_{f_{\tau}}, \theta_{\tau}) \\ \mathcal{L}_{align}(\mathcal{D}_{Q}, G_{aug}; \theta_{f_{\tau}}, \theta_{\tau}) \\ \mathcal{L}_{dist}(\mathcal{D}_{Q}, G_{aug}; \theta_{f_{\tau}}, \theta_{\tau}) \\ \mathcal{L}_{corr}(\mathcal{D}_{Q}, G_{aug}; \theta_{f_{\tau}}, \theta_{\tau}) \end{bmatrix}$$
(15)

where  $\lambda_{miss}, \lambda_{align}, \lambda_{dist}, \lambda_{corr}$  are hyperparameters to be set on.

*Tri-Optimization Based Learning Loop.* At first, we need to learn the parameters  $\theta_{f_{\tau}}$  for the primary task of node classification. Then we learn the surrogate model parameters  $\theta_g$  to measure up the optimal performance of the attacker, which the defense has to match. After this, we learn the augmentation parameters  $\theta_{\tau}$  to step up against the optimized attacker. The tri-optimization objective can be formalized as follows:

$$\begin{aligned} \operatorname{argmin}_{\theta_{\tau}} \mathcal{L}_{defense}(\mathcal{D}_{Q}, G_{aug}; \theta_{g}^{*}, \theta_{f_{\tau}}^{*}, \theta_{\tau}) \\ s.t.\theta_{g}^{*} = \operatorname{argmin}_{\theta_{g}} \mathcal{L}_{attack}(\mathcal{D}_{Q}, G_{aug}; \theta_{g}, \theta_{f_{\tau}}^{*}, \theta_{\tau}) \\ s.t.\theta_{f_{\tau}}^{*} = \operatorname{argmin}_{\theta_{f_{\tau}}} \mathcal{L}_{class}(G_{aug}; \theta_{f_{\tau}}, \theta_{\tau}) \end{aligned}$$
(16)

Function 2 shows the procedure for updating the model parameters. We change the learning step a bit from what was stated before. We update the node augmentation parameters  $\theta_{\tau}$  twice: once after classification (different from Section 4.2.1, "semi"-update) and again at the last step after the surrogate model (as stated in Section 4.2.1, "final"-update). As the node augmentation parameters also take part in the classification task, the loss classification gradients should also be propagated through them. Otherwise, it would downgrade the classification performance of the augmented model. We also pass some epoch-related hyperparameters to run the feature (semi and final) and surrogate updates. Besides, as a learnable feature layer abstraction, we introduce a vector of the same size as number of new nodes × Feature Dimension. The augmented features are always set as zero. So, before passing to the GNN, we add the learnable features to the fixed augmented zero features. Then, they are concatenated with the fixed original node features before being passed to the model. Hence, backpropagation is done during the addition operation of the learnable layer. The whole optimization loop is shown in Figure 2b.

### 4.3 Algorithm: NARGIS

Combining all the modules and functions stated above, we train NARGIS. The training algorithm has been shown in Algorithm 1 (and also for reference in Figure 2a).

### 5 Evaluation

In this section, we describe the datasets, models, attacks and metrics to evaluate our model's performances.

**Function 2:** runTriOptimizationBasedFeatureLearning( $G_{aug}$ ,  $\mathcal{D}_Q$ )

		<i>J</i> .
	Input : Augmented Graph	
	$G_{aug} = (V_{aug}, \mathbf{X}_{aug}, \mathbf{Y}_{aug}, \mathbf{A}_{aug}, n_{aug}, c)$ with no	
	features learned yet;	
	Surrogate Query Edge Dataset $\mathcal{D}_Q$ ;	
	<b>Parameter:</b> number of epochs $\eta_{outer}$ , $\eta_{class}$ , $\eta_{surr}$ , $\eta_{def}$ ;	
	Output :Learned Augmented Graph, $C^{\tau} = (V \mathbf{X}^{\tau} \mathbf{Y} \mathbf{A} \mathbf{n} \mathbf{c})$	
1	$G_{aug} = (Vaug, \mathbf{A}_{aug}, \mathbf{I}_{aug}, \mathbf{A}_{aug}, n_{aug}, c);$ Initiate $\theta^0_{-} \theta^0_{-} \theta^0_{-}$	
	for $t = 0$ to $n = -1$ do	
2	/* Stage 1: Learn Classification Model	*/
3	$\theta_{f_{\tau}}^{0}, \theta_{\tau}^{0} \leftarrow \theta_{f_{\tau}}^{t}, \theta_{\tau}^{t};$	
4	for $t_c = 0$ to $\eta_{class} - 1$ do	
5	Calculate $\mathcal{L}_{class}(G_{aug}; \hat{\theta}_{f_{\tau}}^{t_c}, \hat{\theta}_{\tau}^{t_c})$ using Equation (9);	
6	$\hat{\theta}_{f_{\tau}}^{t_{c}+1}, \hat{\theta}_{\tau}^{t_{c}+1} \leftarrow \text{Update using back-prop from}$	
	$\mathcal{L}_{class}(G_{aug}; \hat{\theta}_{f_{\tau}}^{t_c}, \hat{\theta}_{\tau}^{t_c});$	
7	end	
	/* Feature Learner Semi-Update	*/
8	$\theta_{f_{\tau}}^{t+1}, \theta_{\tau}^{t+2} \leftarrow \theta_{f_{\tau}}^{\prime\prime class}, \theta_{\tau}^{\prime\prime class};$	
9	If $t == 0$ then /* Stage 2: Learn Surrogate Attacker Model	*/
10	$\hat{\theta}^0_a \leftarrow \theta^t_a;$	
11	$\mathbf{for} \ t_s = 0 \ to \ \eta_{surr} - 1 \ \mathbf{do}$	
12	Calculate $\mathcal{L}_{attack}(\mathcal{D}_Q, G_{aug}; \hat{\theta}_g^{t_s}, \theta_{f_\tau}^{t+1}, \theta_{\tau}^{t+\frac{1}{2}})$ usir	ıg
	Equation (10);	
13	$\hat{\theta}_g^{t_s+1} \leftarrow \text{Update using back-prop from}$	
	$\mathcal{L}_{attack}(\mathcal{D}_Q, G_{aug}; \hat{\theta}_g^{t_s}, \theta_{f_\tau}^{t+1}, \theta_{\tau}^{t+\frac{1}{2}});$	
14	end	
	/* Surrogate Attacker Update	*/
15	$\theta_a^{t+1} \leftarrow \hat{\theta}_a^{\eta_{surr}};$	
16	end	
17	else	
18	$\left   \theta_g^{t+1} \leftarrow \theta_g^t \right $	
19	end	
	/* Stage 3 Learn Augmented Features	*/
20	$\hat{\theta}^0_{\tau} \leftarrow \theta^{t+\frac{1}{2}}_{\tau};$	
21	for $t_d = 0$ to $\eta_{def} - 1$ do	
22	Calculate $\mathcal{L}_{defense}(\mathcal{D}_Q, G_{aug}; \theta_g^{t+1}, \theta_f^{t+1}, \hat{\theta}_{\tau}^{t})$ using	
	Equation (15);	
23	$\hat{\theta}_{\tau}^{r_d+1} \leftarrow \text{Update using back-prop from}$	
	$\mathcal{L}_{defense}(\mathcal{D}_Q, G_{aug}; \theta_g^{t+1}, \theta_{f_\tau}^{t+1}, \hat{\theta}_{\tau}^{t_d});$	
24	end	
	/* Feature Learner Final Update	*/
25	$\theta_{\tau}^{t+1} \leftarrow \hat{\theta}_{\tau}^{\prime\prime def};$	
26	end	
	/* $ heta_{ au}^{\eta_{outer}}$ is the learned augmented node features	*/
27	$\mathbf{X}_{aug}^{\tau} \leftarrow \text{replace } \mathbf{X}_{new} \text{ from } \mathbf{X}_{aug} \text{ in Equation (7) with } \theta_{\tau}^{\eta_{outer}};$	
28	return $G_{aug}^{\tau} = (V_{aug}, X_{aug}^{\tau}, Y_{aug}, A_{aug}, n_{aug}, c)$	

### 5.1 Experimental Setup

5.1.1 Datasets. We evaluate our models and attacks on three popular Graph datasets: Cora, Citeseer, Pubmed [25]. These datasets are citation datasets used for benchmarking GNN models [7]. The nodes of the datasets represent publications, and links denote their citations. Cora and Citeseer node features are 0/1 vectors representing the absence/presence of a particular word or tag in a fixed How Feasible is Augmenting Fake Nodes with Learnable Features as a Counter-strategy against Link Stealing Attacks?

ŀ	Algorithm 1: NARGIS							
	<b>Input</b> :Graph $G = (V, \mathbf{X}, \mathbf{Y}, \mathbf{A}, n, c);$							
	<b>Parameter</b> : Number of new nodes to augment, <i>n<sub>new</sub></i> ;							
	Set Threshold $\zeta$ ;							
	number of epochs $\eta_{outer}$ , $\eta_{class}$ , $\eta_{surr}$ , $\eta_{def}$ ;							
	<b>Output</b> :Perturbed Node Posteriors, $\Phi \in [0, 1]^{n \times c}$							
	/* Augmentation Module-Function 1 *,	/						
1	Augmented Graph, $G_{aug} \leftarrow getAugmentedGraph(G);$							
2	<sup>2</sup> Training Edge Dataset, $\mathcal{D}_O \leftarrow$ Edges, sampled negative edges and							
	edge labels from $(G)$ ;							
	<pre>/* Tri-Optimization based Feature Learning</pre>							
	Module-Function 2 */	/						
3	Augmented Graph with Learned Features, $G_{aua}^{\tau} \leftarrow$							
	runTriOptimizationBasedFeatureLearning $(G_{aua}, \mathcal{D}_{O});$							
	/* Get Perturbed Posteriors from the Defense ingrained							
	GNN */	/						
4	$\Phi_{aua} \leftarrow f_{\tau}(G_{aua}^{\tau});$							
	/* Return only the main graph part *	/						
5	Perturbed (Main Graph) Node Posteriors. $\Phi \leftarrow \Phi_{aug}[:n,:]$ :							
6	return $\phi$							

dictionary. Pubmed features are weighted TF-IDF vectors for a vocabulary.

Dataset Configuration for Our Learning: For training each dataset for node classification, we have followed the train, validation, test split settings from [7], where (for example,) for training the Cora Dataset only 20 nodes per class were used, whereas the dataset has 2,708 nodes. While training NARGIS and evaluating the attacks, where the edges form an important part of the training, we at first split the graph based on 70-10-20% train-validation-split of the nodes, and then on the split graph, perform edge split on the positive edges with the same ratio. As stated in [5, 7], we sampled the same number of negative edges for each split. We trained NARGIS on the train edges (positive and negative), validated on validation edges (positive and negative), and tested the attacks on test split edges (positive and negative). All the graphs are trained on transductive settings, where some of the nodes are labeled and training is done on them, while node classification is done on the rest unlabeled nodes.

### 5.2 Software and Hardware Settings

We have used Pytorch ([13]), Pytorch Geometric ([2]) and NetworkX ([3]) for the implementation. We ran our experiments on a server with 3 NVIDIA GeForce RTX 3090 GPUs, totaling 72 GB.

# 5.3 Node Classification GNN models and hyperparameters

As GNN node classification model, we have used three messagepassing GNNs for evaluation: GCN [7], GAT [16] and SAGE [4]. All of the models have two convolution layers with their respective message-passing mechanisms. The dimension of the convolution layers are 16. We also used dropout layers after each convolution layer with p = 0.5. The first convolution layers had ReLU activation, and the last convolution layer, which is also the output layer, had softmax activation. Every GNN were trained for 200 epochs with ADAM [6] optimizer, with learning rate 0.005 and weight decay rate of  $5 \times 10^{-4}$ . After every 10 epochs, we ran validation to find the validation loss, and the model with the least validation loss was saved.

# 5.4 Cluster Numbers

Previously, in Section 3, we have discussed that the cluster number has to be inversely proportional to the Graph Density. We have calculated the graph density of Cora, Citeseer and Pubmed as 0.00144, 0.000823, 0.000228, respectively. Their inverse ratios are: 1 : 1.75 : 6.3 = 10 : 17.5 : 63. So we set cluster numbers (nearby rounded as multiples of 10) 10, 20, 60 for Cora, Citeseer and Pubmed, respectively.

# 5.5 Tri-Optimization Based Module's Hyperparameters

For every GNN model with basic (unguarded) settings, the corresponding augmented GNN model also had the same configuration and convolution layers. The number of epochs  $\eta_{outer}$ ,  $\eta_{class}$ ,  $\eta_{def}$ in Function 2 in Section 4, are set as 10, 200, 50. For surrogate learning, we set the batch size of 512 for training edge set and set  $\eta_{surr}$ as the number of epochs needed to learn the whole set of training edges. Classification, surrogate, and defense Learning models are optimized using Adam [6] with learning rate 0.01, 0.001 and 0.001, respectively. Both classification and defense learning have weight decay of  $5 \times 10^{-4}$ . The surrogate attacker model is a learnable matrix of dimension *class number* × *hidden dimension* (set as 10). We multiply the posteriors from the endpoints with the matrix to get two vectors, and then we take the sigmoid of their dot product. The loss hyperparameters  $\lambda_{misc}$ ,  $\lambda_{align}$ ,  $\lambda_{dist}$  and  $\lambda_{corr}$  are set as 4, 0.8, 2 and 0.6 through grid search, respectively.

### 5.6 Attack Models

For the Attack Models, we have used the same settings as described in [5], for supervised attacker, reference models, shadow GNN and shadow reference models. The supervised attacker Model is an MLP with three hidden layers of 32 dimension, which is trained for 50 epochs with learning rate 0.005 with Adam. Both the reference and shadow reference model for the Attacker are MLPs with two hidden layers with 64 and 32 nodes, which is trained for 50 epochs with Adam optimizer of learning rate 0.01 and weight decay as same. The main and shadow reference models are used for the formation of features for the edges from the posteriors of the nodes from the main and shadow dataset, respectively. The shadow GNN model for learning on shadow dataset is a GCN, which has the same architecture as the classification model, except the hidden dimension is 32 and it is trained for 100 epochs. Shadow GNN is trained to apply transfer learning in four forms of attacks (Attack-1, 4, 5 and 7).

## 5.7 How to Interpret the Experiments

Combining Tables 3 and 4, we have discussed about ten tasks: (1) Node prediction, (2-9) eight forms of Link Stealing Attacks, (10) LinkTeller Attack. For the first task, we want the performance under the defense model to be decreased as low as possible. For the rest of the nine tasks, we want the performance of the attack against the defense model to be decreased as much as possible. A good defensive model has higher node prediction accuracy (indicating lower fidelity loss) and lower AUC (higher attack thwarted).

We also calculated a loss concerning each metric's Baseline (Undefended) model. For accuracy, the loss w.r.t. the baseline model should be as low as possible. For AUC, the loss w.r.t. the baseline model should be as high as possible.

The models marked with a long cross (†) are tuned for the highest fidelity/node prediction accuracy. So, their tradeoff is lower privacy (Higher AUC). In contrast, the models marked with an asterisk (\*) are tuned for the highest privacy (lower AUC), where the tradeoff is possibly lower fidelity.

### 5.8 Metrics

For the node prediction task, we have used accuracy as our performance metric. For link stealing task evaluation, as per the works of [5, 9, 22, 30], we have used AUC (Area under ROC Curve) metric. This metric is used for two reasons: (1) it is safe from class imbalance issue, unlike precision or recall (2) For unsupervised classifications where no labels are present, and we have to decide the label comparing with a threshold, AUC can denote the probability that, under any set threshold, a randomly selected positive edge node pair will have higher probability than a randomly selected negative edge node pair. If all node pairs are ranked randomly, AUC will be 0.5. Each evaluation were run three times with same seed for reproducibility, and the mean value was reported. Also, we have reported the accuracy and AUC loss compared to the Basic (Undefended) model. Moreover, as Attack-0 settings have no knowledge available for the attackers, it is considered as the most trivial form of attack. Hence, our tunings were done considering this trivial attack's performance, as the models are at least expected to defend Attack-0, the most trivial one.

### 5.9 Baseline Models

For baseline evaluation, we have chosen two differential privacy (DP) based defense models, EdgeRand and LapGraph, from [22]. We have chosen three versions of them, with varying DP parameter  $\epsilon$  (lower value indicates higher privacy, higher value indicates higher fidelity): *EdgeRand* ( $\epsilon = 6$ ), *LapGraph* ( $\epsilon = 6$ ) and *LapGraph*  $(\epsilon = 0.1)$ . It is to be noted that, as these models have trade-offs between fidelity and privacy, for the sake of comparison we will consider both cases. In each configuration of a particular GNN model and a dataset, the DP model marked with (†) is the DP model with highest prediction accuracy (denoted onwards as DP-HFP: DP Defense with Highest Fidelity Preferred). Also, the DP model marked with (\*) is the one with lowest Attack-0 AUC (denoted onwards as DP-HPP: DP Defense with Highest Privacy Preferred). For example, in Table 3 for Cora dataset with SAGE GNN model, *EdgeRand* ( $\epsilon = 6$ ) is the *DP-HFP* and *LapGraph* ( $\epsilon = 0.1$ ) is the DP-HPP.

### 5.10 Tuned NARGIS

Considering fidelity-privacy tradeoff, we have evaluated two more versions of NARGIS whose  $\lambda_{align}$  are different from the original one. Whereas the original NARGIS's hyperparameters were tuned for highest fidelity, *NARGIS-DefTuned*'s  $\lambda_{align}$  is adjusted to match the concerning *DP-HPP*'s Attack-0 AUC performance. Additionally,

*NARGIS-PredTuned*'s  $\lambda_{align}$  is adjusted to match the concerning *DP-HFP*'s Prediction performance. They are also marked in the tables with (\*) and (†), respectively. The  $\lambda_{align}$  for each dataset and GNN combinations are described in Table 2.

**Table 2: Tuned Values for**  $\lambda_{align}$  **for** NARGIS-**PredTuned and** NARGIS-**DefTuned** 

GNN Dataset	λ <i>align</i> tuned for Highest Node Prediction NARGIS-PredTuned)	λ <sub>align</sub> tuned for Lowest Attack:0 AUC (NARGIS-DefTuned)
SAGE Cora	0.3	0.25
SAGE Citeseer	0.001	0.005
SAGE Pubmed	0.001	0.002
-		
GCN Cora	2.0	1.0
GCN Citeseer	0.5	0.001
GCN Pubmed	0.15	0.1
GAT Cora	0.15	0.25
GAT Citeseer	0.002	0.001
GAT Pubmed	0.001	0.002

# 5.11 Performance Criteria

We have considered the Node Prediction Performance (Accuracy) and the AUC of all the attacks (0-7) stated in [5] (further described in Table 1) and LinkTeller attack from [22]. We have shown Node prediction, Attack-0, 2, 3, 6; and LinkTeller performances together in Table 3, as there is no shadow dataset involved. For shadow dataset based attacks (Attack- 1,4,5,7), we have shown the results in Table 4. Moreover, as stated in [5], Attack-0,2,3,6 had different feature design schemes. For comparison, we chose the best feature design for each attacks: Correlation distance, Posterior Distance, all features combined and all features combined (again), respectively.

# 5.12 Performance Analysis: Node Prediction

For almost all dataset and GNN combinations, best node prediction accuracies were achieved by NARGIS. Besides, the best accuracy performances among other defenses in these combinations were achieved by the tuned-versions of NARGIS. In GCN:Citeseer combination, the highest accuracy was achieved by *NARGIS-PredTuned*. It is to be noted that only in GAT:Citeseer, the second best performance was achieved by a *DP-HFP*.

# 5.13 Performance Analysis: Singular Attacks (Attack= 0, 2, 3, 6)

For Attack-0, in all datasets, NARGIS's tuned variants have achieved best performance for SAGE; but for GCN and GAT, it is the *DP*-*HPP* who defended the best. In case of Attack-2, the finding is the same ([5] used the correlation difference between posteriors as the best feature for Attack-2 which is same as Attack-0, hence the same performance). Moreover, for Attack-3, none of the variants of NARGIS has been able to beat *DP*-*HPP*. Finally, for Attack-6, *DP*-*HPP* has beaten others in most combinations, while being the second best in cases where other DP based methods got the best result. At first glance, it might seem that NARGIS or its variants are not a good form of defense. But in hindsight, while considering the node prediction accuracy (fidelity), they pose as very balanced defense methods. For most of the cases where NARGIS and its How Feasible is Augmenting Fake Nodes with Learnable Features as a Counter-strategy against Link Stealing Attacks?

variants came second or third-best, their defense performance were not that far, and the fidelities were better.

### 5.14 Performance Analysis: LinkTeller

For LinkTeller ([22]), best performances were achieved by the DP defenses designed to beat it. NARGIS or its variants could not defend it better as LinkTeller focuses on the nodes' influence on each other more rather than exploiting their posterior proximities. As our model has not included inter-node influence-based losses in its design, it has not been able to show near-optimal performance for LinkTeller.

# 5.15 Performance Analysis: Transfer Attacks (Attack- 1, 4, 5, 7)

For SAGE based GNN model, NARGIS's tuned versions have shown superior performances for all dataset and shadow dataset combinations for Attack-1,4,5,7. Even in Pubmed  $\rightarrow$  Citeseer combination for Attack-7, the performance of tuned NARGIS's are close to the *DP-HPP* one. But for the GCN-based GNN model, it is always the DP-based models that show the best defensive performances. For GAT-based models, the performance is almost similar, except for some optimal performances for Attack-7.

# 6 Discussion

In this work, we have introduced a Graph Modification attack as a defensive measure against Link Stealing Attack, In the bigger picture, we investigated whether a form of attack (i.e., Modification) can be introduced as a defensive measure for another form of attack (i.e., inference). Unlike the previous DP-based approaches [22], where the privacy of the graph was ensured through noise addition in the adjacency matrix, i.e., modifying the edge set, our work modified (addition) in the node-set through a clustering and learning based approach. The approach was focused on keeping up the fidelity even with higher privacy, unlike DP-based approaches where there are trade-offs. From the performance analysis on different combinations of GNN models, datasets, attack methods, DP-based Defenses, NARGIS and its variants' performances, we could figure out some key findings:

- NARGIS and its variants have optimal performances mostly for SAGE, and never for GCN.
- NARGIS and its variants, despite being the second best in terms of defending in most cases, are actually the most balanced form of fidelity-privacy trade-off, as they show better accuracy performances for near-equal defensive performances.
- NARGIS can preserve model fidelity better than DP-based defenses for almost similar range of defense performance.

The limitations are:

- Not having optimal performances ubiquitous for all GNN models
- Approaches are justified through the lens of experiments and heuristics rather than theoretical proof
- Unlike for similarity-based attacks, the performances on mutual influence-based LinkTeller Attack are not adequately defended, as the model considers the posterior similarity

Table 3: Defence Performances on Node Prediction and all singular Attacks (Attack-0, 2, 3, 6; LinkTeller). DP-based Defense and NARGIS Models marked with † and \* are tuned, respectively, for high utility (Higher Node Prediction Accuracy) and high privacy (lower Attack-0 AUC).

	Dataset					Cor	a						
GNN	Defense	N Pred	ode iction	Att Corr	ack-0: relation	Atta Post	ack-2: terior	Atta	ack-3: All	Att	ack-6: All	Linl	cteller
		Acc↑	Loss↓	AUC	Loss↑	AUC	Loss↑	AUC	Loss↑	AUC	Loss↑	AUC	Loss↑
SAGE	Basic (†) EdgeRand( $\epsilon = 6$ ) (*) LapGraph( $\epsilon = 0.1$ ) LapGraph( $\epsilon = 6$ ) NARGIS (*)NARGIS-DefTuned (†) NARGIS-PredTuned	0.787 0.564 0.417 0.399 0.81 0.513 0.604	0.223 0.37 0.388 -0.023 0.274 0.183	0.92 0.81 0.7 0.71 0.86 0.65 0.69	0.112 0.226 0.213 0.067 <b>0.272</b> 0.23	0.92 0.81 0.7 0.71 0.86 0.65 0.69	0.112 0.226 0.213 0.067 <b>0.272</b> 0.23	0.93 0.83 0.72 0.74 0.86 0.76 0.79	- 0.096 0.207 0.185 0.069 0.168 0.14	0.93 0.84 0.79 0.8 0.88 0.81 0.82	- 0.087 0.138 0.124 0.052 0.113 0.102	0.98 0.5 0.5 0.61 0.97 0.96 0.96	- 0.476 0.474 0.369 0.01 0.013 0.013
GCN	Basic (†) EdgeRand ( $\epsilon = 6$ ) (*)LapGraph ( $\epsilon = 0.1$ ) LapGraph ( $\epsilon = 6$ ) NARGIS (*) NARGIS-DerTuned (†) NARGIS-PredTuned	0.793 0.64 0.144 0.302 0.852 0.763 0.746	- 0.153 0.649 0.491 <b>-0.059</b> 0.03 0.047	0.94 0.88 0.52 0.67 0.9 0.89 0.89	- 0.057 0.416 0.272 0.039 0.047 0.05	0.94 0.88 0.52 0.67 0.9 0.89 0.89	- 0.057 0.416 0.272 0.039 0.047 0.05	0.93 0.89 0.54 0.69 0.89 0.88 0.89	- 0.044 0.39 0.245 0.037 0.05 0.044	0.93 0.86 0.75 0.79 0.9 0.89 0.9	- 0.071 0.179 0.137 0.03 0.034 0.033	0.93 0.5 0.5 0.59 0.82 0.79 0.78	- 0.43 0.425 0.336 0.109 0.139 0.146
GAT	Basic (†) EdgeRand( $\epsilon = 6$ ) (*) LapGraph( $\epsilon = 0.1$ ) LapGraph( $\epsilon = 6$ ) NARGIS (*) NARGIS-DefTuned (†)NARGIS-PredTuned	0.748 0.399 0.125 0.217 0.71 0.329 0.289	0.349 0.623 0.531 <b>0.038</b> 0.419 0.459	0.92 0.73 0.52 0.61 0.81 0.65 0.63	- 0.188 0.4 0.307 0.105 0.264 0.286	0.92 0.73 0.52 0.61 0.81 0.65 0.63	- 0.188 0.4 0.307 0.105 0.264 0.286	0.91 0.74 0.54 0.62 0.83 0.74 0.72	0.174 0.368 0.288 0.078 0.17 0.186	0.92 0.8 0.75 0.77 0.87 0.87 0.84 0.83	- 0.111 0.165 0.146 0.044 0.081 0.086	0.8 0.5 0.51 0.59 0.71 0.61 0.66	<b>0.296</b> 0.289 0.209 0.089 0.183 0.141

	Dataset	Citeseer											
	Basic	0.687	-	0.93	-	0.93	-	0.94		0.94	-	0.99	-
	$EdgeRand(\epsilon = 6)$	0.5	0.187	0.8	0.123	0.8	0.123	0.82	0.123	0.86	0.084	0.5	0.489
	$LapGraph(\epsilon = 0.1)$	0.547	0.14	0.76	0.171	0.76	0.171	0.77	0.174	0.82	0.122	0.51	0.482
SAGE	$(*)(†)$ LapGraph( $\epsilon = 6$ )	0.553	0.134	0.75	0.182	0.75	0.182	0.76	0.181	0.82	0.12	0.57	0.418
	NARĜIS	0.584	0.103	0.75	0.178	0.75	0.178	0.83	0.111	0.86	0.081	0.97	0.015
	(*) NARGIS-DefTuned	0.495	0.192	0.66	0.266	0.66	0.266	0.81	0.132	0.84	0.102	0.97	0.015
	(†) NARGIS-PredTuned	0.56	0.127	0.77	0.159	0.77	0.159	0.88	0.058	0.89	0.054	0.94	0.054
	Basic	0.679	-	0.95	-	0.95	-	0.94	-	0.95	-	0.95	-
	(†)EdgeRand( $\epsilon = 6$ )	0.46	0.219	0.85	0.102	0.85	0.102	0.85	0.09	0.86	0.092	0.5	0.446
	(*)LapGraph( $\epsilon = 0.1$ )	0.194	0.485	0.53	0.423	0.53	0.423	0.55	0.396	0.8	0.148	0.52	0.431
GCN	$LapGraph(\epsilon = 6)$	0.262	0.417	0.64	0.31	0.64	0.31	0.65	0.29	0.81	0.136	0.57	0.373
	NARGIS	0.463	0.216	0.82	0.127	0.82	0.127	0.86	0.088	0.9	0.047	0.86	0.089
	(*)NARGIS-DefTuned	0.185	0.494	0.71	0.24	0.71	0.24	0.77	0.178	0.84	0.109	0.84	0.103
	(†) NARGIS-PredTuned	0.577	0.102	0.9	0.054	0.9	0.054	0.9	0.043	0.92	0.03	0.84	0.11
	Basic	0.677	-	0.93	-	0.93	-	0.92	-	0.93	-	0.87	-
	(†) EdgeRand( $\epsilon = 6$ )	0.3	0.377	0.68	0.249	0.68	0.249	0.7	0.222	0.83	0.1	0.5	0.366
	(*)LapGraph( $\epsilon = 0.1$ )	0.225	0.452	0.55	0.387	0.55	0.387	0.56	0.36	0.8	0.132	0.5	0.363
GAT	$LapGraph(\epsilon = 6)$	0.286	0.391	0.65	0.286	0.65	0.286	0.65	0.27	0.81	0.118	0.58	0.286
	NARGIS	0.626	0.051	0.89	0.045	0.89	0.045	0.9	0.025	0.91	0.017	0.77	0.097
	(*)NARGIS-DefTuned	0.238	0.439	0.68	0.253	0.68	0.253	0.8	0.124	0.88	0.048	0.66	0.208
	(†) NARGIS-PredTuned	0.295	0.382	0.74	0.189	0.74	0.189	0.83	0.091	0.9	0.03	0.61	0.254

	Dataset		Pubmed										
	Basic	0.767	-	0.86	-	0.86	-	0.9	-	0.9	-	0.99	-
	(†) EdgeRand( $\epsilon = 6$ )	0.705	0.062	0.76	0.107	0.76	0.107	0.77	0.13	0.81	0.092	0.5	0.49
	$LapGraph(\epsilon = 0.1)$	0.7	0.067	0.75	0.11	0.75	0.11	0.77	0.132	0.81	0.092	0.5	0.488
SAGE	$(*)$ LapGraph( $\epsilon = 6$ )	0.7	0.067	0.75	0.113	0.75	0.113	0.77	0.135	0.8	0.094	0.5	0.489
	NARGIS	0.734	0.033	0.82	0.039	0.82	0.039	0.87	0.033	0.87	0.031	0.98	0.011
	(*)NARGIS-DefTuned	0.718	0.049	0.71	0.148	0.71	0.148	0.8	0.105	0.83	0.066	0.97	0.017
	(†) NARGIS-PredTuned	0.578	0.189	0.71	0.151	0.71	0.151	0.79	0.111	0.83	0.064	0.96	0.032
	Basic	0.785	-	0.87	-	0.87	-	0.89	-	0.86	-	0.93	-
	$EdgeRand(\epsilon = 6)$	0.407	0.378	0.61	0.26	0.61	0.26	0.68	0.206	0.77	0.091	0.5	0.432
	(*)LapGraph( $\epsilon = 0.1$ )	0.429	0.356	0.51	0.36	0.51	0.36	0.52	0.372	0.78	0.087	0.5	0.432
GCN	(†) LapGraph( $\epsilon = 6$ )	0.458	0.327	0.56	0.311	0.56	0.311	0.58	0.312	0.78	0.084	0.53	0.399
	NARGIS	0.672	0.113	0.8	0.068	0.8	0.068	0.85	0.039	0.86	0.007	0.85	0.081
	(*)NARGIS-DefTuned	0.545	0.24	0.57	0.296	0.57	0.296	0.84	0.051	0.88	-0.011	0.82	0.107
	(†) NARGIS-PredTuned	0.58	0.205	0.55	0.317	0.55	0.317	0.83	0.061	0.88	-0.015	0.8	0.129
	Basic	0.769	-	0.85	-	0.85	-	0.89	-	0.89	-	0.9	-
	(*)EdgeRand( $\epsilon = 6$ )	0.413	0.356	0.51	0.347	0.51	0.347	0.5	0.389	0.78	0.107	0.5	0.398
	$LapGraph(\epsilon = 0.1)$	0.408	0.361	0.52	0.335	0.52	0.335	0.53	0.36	0.78	0.11	0.5	0.397
GAT	(†) LapGraph( $\epsilon = 6$ )	0.452	0.317	0.57	0.283	0.57	0.283	0.58	0.308	0.79	0.103	0.53	0.366
	NARGIS	0.657	0.112	0.8	0.052	0.8	0.052	0.86	0.032	0.86	0.026	0.82	0.08
	(*) NARGIS-DefTuned	0.65	0.119	0.68	0.17	0.68	0.17	0.74	0.151	0.83	0.06	0.72	0.174
	(†) NARGIS-PredTuned	0.641	0.128	0.68	0.178	0.68	0.178	0.74	0.15	0.84	0.054	0.71	0.19

Table 4: Defence Performances on all transfer Attacks (Attack-1, 4, 5, 7). DP-based Defense and NARGIS Models marked with † and \* are tuned, respectively, for high utility (Higher Node Prediction Accuracy) and high privacy (lower Attack-0 AUC).

Data	set→Shadow Dataset	$Cora \rightarrow Citeseer$									$Cora \rightarrow Pubmed$						
GNN	Defense	Atta	ck-1	Atta	ck-4	Atta	ck-5	Atta	ck-7	Atta	ck-1	Atta	ck-4	Atta	ck-5	Atta	ck-7
		AUC↓	Loss↑	AUC↓	Loss↑	AUC↓	Loss↑	AUC↓	Loss↑	AUC↓	Loss↑	AUC↓	Loss↑	AUC↓	Loss↑	AUC↓	Loss↑
SAGE	$\begin{array}{l} \text{Basic} \\ (\dagger)\text{EdgeRand}(\epsilon=6) \\ (\ast)\text{LapGraph}(\epsilon=0.1) \\ \text{LapGraph}(\epsilon=6) \\ \text{NARGIS} \\ (\ast)\text{NARGIS-DefTuned} \\ (\dagger)\text{NARGIS-PredTuned} \end{array}$	0.92 0.81 0.7 0.7 0.82 0.63 0.68	0.1 0.22 0.21 0.09 <b>0.28</b> 0.24	0.91 0.82 0.7 0.7 0.81 0.61 0.64	0.1 0.21 0.21 0.1 <b>0.3</b> 0.27	0.93 0.8 0.7 0.7 0.87 0.7 0.7 0.75	0.12 0.23 0.22 0.05 0.23 0.18	0.92 0.82 0.74 0.75 0.85 0.75 0.75	0.1 0.18 0.17 0.07 0.17 0.18	0.78 0.71 0.64 0.64 0.77 0.58 0.62	- 0.08 0.14 0.14 0.02 <b>0.2</b> 0.16	0.85 0.75 0.66 0.66 0.78 0.59 0.63	0.1 0.19 0.19 0.07 <b>0.26</b> 0.22	0.73 0.71 0.7 0.7 0.74 0.69 0.7	- 0.02 0.03 0.03 -0.01 <b>0.04</b> 0.03	0.86 0.76 0.73 0.72 0.81 0.72 0.74	- 0.1 0.13 <b>0.14</b> 0.05 <b>0.14</b> 0.12
GCN	$\begin{array}{c} \text{Basic} \\ (\dagger)  \text{EdgeRand}(\epsilon=6) \\ (\ast) \text{LapGraph}(\epsilon=0.1) \\ \text{LapGraph}(\epsilon=6) \\ \text{NARGIS} \\ (\ast) \text{NARGIS-DefTuned} \\ (\dagger) \text{NARGIS-PredTuned} \end{array}$	0.93 0.85 0.51 0.66 0.89 0.88 0.88	- 0.09 0.42 0.28 0.04 0.05 0.06	0.93 0.88 0.52 0.66 0.89 0.87 0.87	- 0.06 0.42 0.27 0.05 0.06 0.06	0.91 0.82 0.53 0.65 0.88 0.88 0.88	- 0.09 0.38 0.26 0.03 0.03 0.03	0.92 0.85 0.7 0.74 0.88 0.88 0.88	- 0.07 0.22 0.18 0.04 0.05 0.05	0.86 0.83 0.52 0.62 0.85 0.84 0.84	- 0.03 0.23 0.01 0.02 0.02	0.9 0.85 0.52 0.64 0.86 0.85 0.85	- 0.05 0.38 0.26 0.04 0.05 0.05	0.76 0.72 0.67 0.69 0.77 0.76 0.76	- 0.04 0.09 0.07 -0.01 0 0	0.9 0.86 0.72 0.71 0.88 0.86 0.86	- 0.03 0.18 <b>0.19</b> 0.02 0.04 0.04
GAT	Basic (†)EdgeRand( $\epsilon = 6$ ) (*)LapGraph( $\epsilon = 0.1$ ) LapGraph( $\epsilon = 6$ ) NARGIS (*)NARGIS-DefTuned (†)NARGIS-PredTuned	$\begin{array}{c} 0.9\\ 0.72\\ 0.51\\ 0.6\\ 0.72\\ 0.56\\ 0.64 \end{array}$	0.18 0.39 0.3 0.18 0.34 0.26	0.9 0.72 0.52 0.6 0.79 0.72 0.68	0.18 0.38 0.3 0.11 0.18 0.22	0.91 0.74 0.55 0.6 0.84 0.73 0.71	0.17 0.36 0.31 0.07 0.18 0.2	0.9 0.77 0.71 0.72 0.84 0.77 0.77	0.13 0.19 0.18 0.07 0.13 0.14	0.73 0.64 0.52 0.57 0.68 0.57 0.61	0.1 0.21 0.17 0.05 0.16 0.12	0.84 0.67 0.52 0.57 0.7 0.55 0.63	0.16 0.32 0.26 0.13 0.28 0.2	0.72 0.7 0.67 0.69 0.7 0.63 0.64	0.03 0.05 0.03 0.02 <b>0.09</b> 0.09	0.84 0.7 0.67 0.66 0.74 0.66 0.65	0.14 0.16 0.18 0.1 0.17 0.19
Data	set→Shadow Dataset			Ci	iteseeı	: → Co	ra					Cite	eseer –	→ Pubi	ned		
SAGE	Basic EdgeRand( $\epsilon = 6$ ) LapGraph( $\epsilon = 0.1$ ) (†)(*)LapGraph( $\epsilon = 6$ ) NARGIS (*)NARGIS-DefTuned (†)NARGIS-PredTuned	0.91 0.79 0.74 0.74 0.72 0.63 0.72	- 0.12 0.17 0.18 0.19 <b>0.28</b> 0.19	0.91 0.8 0.75 0.74 0.72 0.62 0.72	- 0.11 0.16 0.17 0.2 <b>0.29</b> 0.19	0.93 0.82 0.77 0.77 0.79 0.71 0.82	- 0.11 0.16 0.17 0.14 <b>0.22</b> 0.11	0.94 0.86 0.81 0.81 0.82 0.8 0.8 0.87	- 0.08 0.13 0.13 0.12 <b>0.14</b> 0.07	0.82 0.75 0.71 0.7 0.68 0.61 0.66	- 0.07 0.12 0.13 0.15 <b>0.22</b> 0.17	0.85 0.76 0.72 0.7 0.68 0.61 0.68	- 0.09 0.13 0.15 0.17 <b>0.24</b> 0.17	0.63 0.61 0.62 0.61 0.62 0.62 0.61	- 0.02 0.02 0.02 0.01 0.01 0.02	0.88 0.85 0.81 0.8 0.77 0.73 0.77	- 0.04 0.07 0.08 0.11 <b>0.16</b> 0.11
GCN	$\begin{array}{l} \text{Basic} \\ (\dagger) \operatorname{EdgeRand}(\epsilon=6) \\ (*) \operatorname{LapGraph}(\epsilon=0.1) \\ \operatorname{LapGraph}(\epsilon=6) \\ \operatorname{NARGIS} \\ (*) \operatorname{NARGIS-DefTuned} \\ (\dagger) \operatorname{NARGIS-PredTuned} \end{array}$	0.93 0.82 0.51 0.62 0.77 0.62 0.85	0.11 0.42 0.31 0.16 0.32 0.08	0.94 0.84 0.51 0.63 0.77 0.68 0.86	0.09 0.42 0.31 0.16 0.26 0.08	0.94 0.85 0.56 0.66 0.85 0.76 0.9	0.1 0.38 0.28 0.09 0.18 0.05	0.94 0.88 0.77 0.78 0.9 0.85 0.92	- 0.06 0.17 0.16 0.04 0.09 0.03	0.87 0.82 0.52 0.61 0.72 0.56 0.81	0.05 0.35 0.26 0.15 0.31 0.06	0.89 0.83 0.52 0.61 0.74 0.57 0.82	0.06 0.37 0.28 0.15 0.32 0.07	0.63 0.61 0.6 0.61 0.63 0.63 0.64	- 0.02 0.04 0.02 0 0 -0.01	0.91 0.89 0.76 0.76 0.83 0.76 0.85	- 0.01 0.15 0.08 0.15 0.06
GAT	$\begin{array}{l} \text{Basic} \\ (\dagger) \text{EdgeRand}(\epsilon=6) \\ (\ast) \text{LapGraph}(\epsilon=6) \\ \text{LapGraph}(\epsilon=6) \\ \text{NARGIS} \\ (\ast) \text{NARGIS-DefTuned} \\ (\dagger) \text{NARGIS-PredTuned} \end{array}$	0.89 0.67 0.54 0.63 0.81 0.64 0.63	- 0.22 0.35 0.26 0.08 0.26 0.26	0.9 0.68 0.54 0.63 0.82 0.7 0.66	- 0.22 0.35 0.26 0.08 0.2 0.24	0.92 0.7 0.59 0.66 0.9 0.78 0.81	- 0.22 0.33 0.26 0.02 0.14 0.11	0.93 0.81 0.77 0.79 0.91 0.86 0.87	- 0.12 0.16 0.14 0.02 0.07 0.06	0.78 0.62 0.52 0.59 0.76 0.61 0.59	0.16 0.26 0.19 0.02 0.18 0.19	0.84 0.63 0.52 0.6 0.78 0.61 0.63	- 0.21 0.32 0.24 0.06 0.23 0.21	0.61 0.6 0.63 0.61 0.61 0.61	- 0.01 0.01 -0.01 0.01 0 0.01	0.86 0.78 0.76 0.76 0.81 0.75 0.78	- 0.08 0.11 0.1 0.06 <b>0.11</b> 0.08
Data	set→Shadow Dataset			Р	ubmed	l → Co	ra		Pubmed→ Citeseer								
SAGE	$\begin{array}{l} \text{Basic} \\ (\dagger) \text{EdgeRand}(\epsilon=6) \\ \text{LapGraph}(\epsilon=0.1) \\ (\ast) \text{LapGraph}(\epsilon=6) \\ \text{NARGIS} \\ (\ast) \text{NARGIS-DefTuned} \\ (\dagger) \text{NARGIS-PredTuned} \end{array}$	0.87 0.76 0.75 0.75 0.81 0.72 0.67	0.11 0.12 0.12 0.06 0.15 0.2	0.88 0.76 0.76 0.76 0.84 0.74 0.75	0.12 0.12 0.12 0.04 <b>0.14</b> 0.14	0.84 0.72 0.72 0.69 0.79 0.69 0.65	0.12 0.12 0.15 0.05 0.15 <b>0.19</b>	0.91 0.84 0.84 0.84 0.88 0.85 0.85	- 0.07 0.06 0.07 0.03 0.06 0.05	0.88 0.76 0.75 0.75 0.81 0.72 0.67	0.12 0.12 0.12 0.06 <b>0.15</b> 0.2	0.89 0.76 0.76 0.76 0.85 0.74 0.76	0.12 0.12 0.13 0.04 <b>0.14</b> 0.13	0.89 0.77 0.78 0.76 0.85 0.75 0.75	0.12 0.1 0.13 0.04 0.13 <b>0.14</b>	0.9 0.82 0.83 0.82 0.87 0.83 0.85	<b>0.08</b> 0.08 <b>0.08</b> 0.03 0.07 0.05
GCN	$\begin{array}{c} \text{Basic} \\ \text{EdgeRand}(\epsilon=6) \\ (*)\text{LapGraph}(\epsilon=0.1) \\ (\dagger)\text{LapGraph}(\epsilon=6) \\ \text{NARGIS} \\ (*)\text{NARGIS-DefTuned} \\ (\dagger) \text{NARGIS-PredTuned} \end{array}$	0.83 0.62 0.51 0.54 0.81 0.52 0.55	0.2 0.32 0.28 0.02 0.3 0.28	0.89 0.6 0.51 0.57 0.84 0.73 0.71	0.29 0.38 0.32 0.05 0.16 0.19	0.81 0.78 0.55 0.5 0.75 0.55 0.55	- 0.03 0.26 <b>0.31</b> 0.06 0.26 0.26	0.92 0.88 0.86 0.85 0.9 0.92 0.92	- 0.04 0.05 <b>0.07</b> 0.02 0 0	0.81 0.63 0.51 0.54 0.79 0.53 0.54	0.18 0.3 0.27 0.03 0.28 0.27	0.89 0.56 0.51 0.57 0.84 0.75 0.74	0.33 0.38 0.32 0.05 0.14 0.15	0.82 0.66 0.54 0.61 0.77 0.55 0.56	- 0.16 0.28 0.21 0.05 0.27 0.26	0.91 0.87 0.86 0.86 0.88 0.92 0.92	- 0.04 0.05 0.05 0.02 -0.01 -0.01
GAT	Basic (*)EdgeRand( $\epsilon = 6$ ) LapGraph( $\epsilon = 0.1$ ) (†)LapGraph( $\epsilon = 6$ ) NARGIS (*) NARGIS-DefTuned (*)NARGIS-ProdTuned	0.85 0.5 0.51 0.56 0.8 0.64 0.63	- 0.34 0.33 0.28 0.05 0.21 0.22	0.87 0.53 0.52 0.57 0.82 0.68 0.67	- 0.34 0.35 0.29 0.04 0.19 0.2	0.82 0.72 0.55 0.52 0.77 0.63 0.62	0.1 0.27 0.3 0.05 0.19 0.2	0.9 0.88 0.87 0.87 0.87 0.87 0.84 0.85	- 0.01 0.02 0.02 0.03 <b>0.05</b>	0.85 0.5 0.51 0.57 0.8 0.64 0.63	- 0.35 0.34 0.29 0.05 0.21 0.22	0.87 0.53 0.52 0.57 0.83 0.69 0.68	- 0.34 0.35 0.3 0.04 0.18 0.18	0.87 0.7 0.56 0.62 0.84 0.72 0.72	- 0.17 0.3 0.25 0.03 0.15 0.15	0.89 0.88 0.86 0.86 0.86 0.85 0.85	- 0.01 0.03 0.03 0.03 <b>0.04</b> 0.03

measures only- not the mutual influence between node features.

Based on the findings, we propose some investigation scenarios and hypotheses:

Formalizing the Influence of  $\lambda$  Weights in Defense Loss. While

tuning the model for best performances, the main adjustment was done for the loss weights  $\lambda_{misc}$ ,  $\lambda_{align}$ ,  $\lambda_{dist}$  and  $\lambda_{corr}$ . Their corresponding loss gradients for the output layer have closed-form which could open the door for the theoretical analysis of what should be the tuning mechanism of these loss weights to achieve optimum performance. As the corresponding losses involve contrasting objectives, proper tuning scheme can help achieve the desired fidelity-privacy tradeoff.

 $\lambda_{align}$  for Tuning Our heuristic for using alignment loss weight as the tuning parameter for comparison against *DP-HPP* and *DP-HFP* was that this loss tries to keep the model fidelity performance up when the other losses are focusing on perturbing the posterior spaces. Jointly tuning one of the other contrasting loss weights along with  $\lambda_{align}$  could help as a trade-off tuning scheme.

**Integrating Node Influence.** NARGIS is focused more on perturbing the model output space through the augmented nodes as the center of the pre-calculated spectral clusters, rather than through the most influential nodes. Apart from the graph topology which was used in computing the clusters, we can also integrate the nodes' influence over others while forming the clusters.

**Message-Passing Mechanism.** Our model is message-passing agnostic, as the main defensive layer is established on the graph input side, rather than in the graph message-passing mechanism, unlike [27]. As we saw SAGE's superior performance over GCN or GAT, it can be a future direction to investigate how to integrate the augmentation scheme inside the message-passing mechanism.

### 7 Related Works

In this section, we will introduce the contemporary works on attacks on graphs and GNNs, and the defensive measures against the attacks. For an in-depth introduction to the message-passing GNNs, we refer to the works [4, 7, 8, 16, 24] to the readers.

### 7.1 Attacks on Graph and GNN Models

Literature have discussed attacks on graph in different settings: poison (manipulating the model through corrupted data), evasion (manipulating the data to bypass the model without corrupting), reconstruction (inferring the data through model posteriors and other information), inversion (inferring the model parameters) etc. Under the evasion settings, Zou et al. [31] has proposed a topological edge selection-based strategy for inserting fake nodes inside a graph, where the features are learned using a surrogate model. Chen et al. [1] have shown that by improving the homophily unnoticeability of the graphs, Graph Injection attacks, another example of evasion setting, can outperform previous homophily-based attacks. Different attacks have been proposed for inferring the graph structures and memberships. He et al. [5] introduced Link stealing attack from the posteriors of GNNs on the basis of proximity-based features. In [22], Wu et al., have evolved from the notion of similarity between node posteriors and leverage node feature perturbation-based influence analysis for predicting graph edges. Zhang et al. [26] has shown that the privacy risks for edge groups from the membership inference attack are uneven and performed group-based attacks depending on the unequal vulnerability. Different works have also discussed poisoning the graph for reducing the node classification performance through Meta-Learning (Meta-attack [32]), Reinforcement Learning (NIPA [15]), Fast Gradient Sign Model linearization (AFGSM [18]) etc. It should be noted that our focus attack is in reconstruction settings. In contrast, our defense is influenced by poisoning settings, as it manipulates the graph structure during training. Defending the model and data from an attack under a

particular setting through adapting another attack settings is the main focus of our work.

## 7.2 Defense against Attack on Graph Models

Several defense mechanisms have been proposed and validated for attack on Graphs and GNNs. The defensive strategies for these works are mostly focused on robust perturbations of Graph embeddings and Differential Privacy guaranteed learning of Graphs. Zhang and Zitnik has introduced GNNGuard [27]- based on neighbor importance estimation and layer-wise graph memory in order to prune possible fake and suspicious edges to stabilize the original graph against the training-time perturbation attack. Differential privacy-based approaches have been a popular method for defending against graph reconstruction (specially link stealing) attacks. Kolluri et al. [9] developed a novel architecture named LPGNet using only multilayer perceptrons to model both the node feature information and some carefully chosen graph structural information from the graph edges after compressing the edge information as a feature vector. In this way, edges are kept separate from the attacks. Another approach called Blink ([30]) injects noise into each node's adjacency list and degree in a decentralized setting and guarantees Localized Differential Privacy; and then uses Bayesian estimation in the server to receive original edge information. Zhou et al. have introduced the Markov chain-based graph reconstruction attack and defense under information theory-guided assumptions [29]. Wu et al., apart from LinkTeller, also introduced in [22] two DP-based defenses named EdgeRand and LapGraph against link inference attacks, as discussed in previous sections. Our work differs from these DP-based reconstruction-defending approaches in different aspects. Firstly, unlike the DP-based approaches, our model does not change the original edges and node features through noise injection. Hence, the originality of the concerned data is preserved. Secondly, we leverage a deterministic approach- spectral clustering, for injecting the nodes in focused places- instead of the randomized approach prioritized by the DP-based models. This can pave the way for more stable and consistent performance analysis and guarantees.

## 8 Conclusion and Future Work

In this paper, we introduce a defense against link-stealing attacks inspired by the paradigm of using attacks as a form of defense. We propose NARGIS - a node augmentation-based defense for GNN, which can restrict the model from providing posteriors leading to edge inference attackers, by perturbing the embedding space through augmenting nodes with learnable features instead of fixed ones. We show that our model can achieve good fidelity-privacy tradeoff in some cases, while there are other scopes to improve on. **Future work.** In the future we want to theoretically find a performance bound - concerning the tunable defense loss weights in the model optimization loop. We also would like to investigate how to make the model performance better across different messagepassing mechanisms.

### References

 Yongqiang Chen, Han Yang, Yonggang Zhang, Kaili Ma, Tongliang Liu, Bo Han, and James Cheng. 2022. Understanding and improving graph injection attack by promoting unnoticeability. arXiv preprint arXiv:2202.08057 (2022).

- [2] Matthias Fey and Jan Eric Lenssen. 2019. Fast graph representation learning with PyTorch Geometric. arXiv preprint arXiv:1903.02428 (2019).
- [3] Aric Hagberg, Pieter J Swart, and Daniel A Schult. 2008. Exploring network structure, dynamics, and function using NetworkX. Technical Report. Los Alamos National Laboratory (LANL), Los Alamos, NM (United States).
- [4] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. Advances in neural information processing systems 30 (2017).
- [5] 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.
- [6] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014).
- [7] Thomas N Kipf and Max Welling. 2016. Semi-Supervised Classification with Graph Convolutional Networks. In International Conference on Learning Representations.
- [8] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. arXiv preprint arXiv:1611.07308 (2016).
- [9] Aashish Kolluri, Teodora Baluta, Bryan Hooi, and Prateek Saxena. 2022. LPGNet: Link private graph networks for node classification. In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. 1813–1827.
- [10] Weiwen Liu, Yin Zhang, Jianling Wang, Yun He, James Caverlee, Patrick PK Chan, Daniel S Yeung, and Pheng-Ann Heng. 2021. Item relationship graph neural networks for e-commerce. *IEEE Transactions on Neural Networks and Learning Systems* 33, 9 (2021), 4785–4799.
- [11] Inc Meta. 2022. Graph API Documentation Meta for Developers. https: //developers.facebook.com/docs/graph-api/overview/ [Accessed: (May 31, 2024)].
- [12] Neo4j, Inc. 2023. Neo4j Aura: The Managed, Cloud Graph Database Service. https://neo4j.com/cloud/aura/ Accessed: 2024-10-13.
- [13] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems 32 (2019).
- [14] Aravind Sankar, Yozen Liu, Jun Yu, and Neil Shah. 2021. Graph neural networks for friend ranking in large-scale social platforms. In *Proceedings of the Web Conference 2021*. 2535–2546.
- [15] Yiwei Sun, Suhang Wang, Xianfeng Tang, Tsung-Yu Hsieh, and Vasant Honavar. 2020. Adversarial attacks on graph neural networks via node injections: A hierarchical reinforcement learning approach. In Proceedings of the Web Conference 2020. 673–683.
- [16] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In International Conference on Learning Representations. https://openreview.net/forum?id=rJXMpikCZ
- [17] Ulrike Von Luxburg. 2007. A tutorial on spectral clustering. Statistics and computing 17 (2007), 395–416.
- [18] Jihong Wang, Minnan Luo, Fnu Suya, Jundong Li, Zijiang Yang, and Qinghua Zheng. 2020. Scalable attack on graph data by injecting vicious nodes. *Data Mining and Knowledge Discovery* 34 (2020), 1363–1389.
- [19] Xiaoyun Wang, Minhao Cheng, Joe Eaton, Cho-Jui Hsieh, and Felix Wu. 2018. Attack graph convolutional networks by adding fake nodes. arXiv preprint arXiv:1810.10751 (2018).
- [20] Wikipedia. Accessed: (May 31, 2024). Jensen-Shannon divergence. https://en. wikipedia.org/wiki/Jensen-Shannon\_divergence.
- [21] Dong-Dong Wu, Chilin Fu, Weichang Wu, Wenwen Xia, Xiaolu Zhang, Jun Zhou, and Min-Ling Zhang. 2024. Efficient Model Stealing Defense with Noise Transition Matrix. (2024).
- [22] Fan Wu, Yunhui Long, Ce Zhang, and Bo Li. 2022. Linkteller: Recovering private edges from graph neural networks via influence analysis. In 2022 ieee symposium on security and privacy (sp). IEEE, 2005–2024.
- [23] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. 2022. Graph neural networks in recommender systems: a survey. *Comput. Surveys* 55, 5 (2022), 1–37.
- [24] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How Powerful are Graph Neural Networks?. In International Conference on Learning Representations.
- [25] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. 2016. Revisiting semisupervised learning with graph embeddings. In *International conference on machine learning*. PMLR, 40–48.
- [26] He Zhang, Bang Wu, Shuo Wang, Xiangwen Yang, Minhui Xue, Shirui Pan, and Xingliang Yuan. 2023. Demystifying uneven vulnerability of link stealing attacks against graph neural networks. In *International Conference on Machine Learning*. PMLR, 41737–41752.
- [27] Xiang Zhang and Marinka Zitnik. 2020. Gnnguard: Defending graph neural networks against adversarial attacks. Advances in neural information processing systems 33 (2020), 9263–9275.
- [28] Zhikun Zhang, Min Chen, Michael Backes, Yun Shen, and Yang Zhang. 2022. Inference attacks against graph neural networks. In 31st USENIX Security Symposium (USENIX Security 22). 4543–4560.

- [29] Zhanke Zhou, Chenyu Zhou, Xuan Li, Jiangchao Yao, Quanming Yao, and Bo Han. 2023. On strengthening and defending graph reconstruction attack with markov chain approximation. arXiv preprint arXiv:2306.09104 (2023).
- [30] Xiaochen Zhu, Vincent YF Tan, and Xiaokui Xiao. 2023. Blink: Link Local Differential Privacy in Graph Neural Networks via Bayesian Estimation. In Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. 2651–2664.
- [31] Xu Zou, Qinkai Zheng, Yuxiao Dong, Xinyu Guan, Evgeny Kharlamov, Jialiang Lu, and Jie Tang. 2021. Tdgia: Effective injection attacks on graph neural networks. In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining. 2461–2471.
- [32] Daniel Zügner, Oliver Borchert, Amir Akbarnejad, and Stephan Günnemann. 2020. Adversarial attacks on graph neural networks: Perturbations and their patterns. ACM Transactions on Knowledge Discovery from Data (TKDD) 14, 5 (2020), 1–31.