LPGNet: Link Private Graph Networks for Node Classification

Aashish Kolluri
aashish7@comp.nus.edu.sg
National University of Singapore
School of Computing
Singapore

Teodora Baluta
teobaluta@comp.nus.edu.sg
National University of Singapore
School of Computing
Singapore

Bryan Hooi
bhooi@comp.nus.edu.sg
National University of Singapore
School of Computing
Singapore

Prateek Saxena
prateeks@comp.nus.edu.sg
National University of Singapore
School of Computing
Singapore

ABSTRACT

Classification tasks on labeled graph-structured data have many important applications ranging from social recommendation to financial modeling. Deep neural networks are increasingly being used for node classification on graphs, wherein nodes with similar features have to be given the same label. Graph convolutional networks (GCNs) are one such widely studied neural network architecture that perform well on this task. However, powerful link-stealing attacks on GCNs have recently shown that even with black-box access to the trained model, inferring which links (or edges) are present in the training graph is practical. In this paper, we present a new neural network architecture called LPGNet for training on graphs with privacy-sensitive edges. LPGNet provides differential privacy (DP) guarantees for edges using a novel design for how graph edge structure is used during training. We empirically show that LPGNet models often lie in the sweet spot between providing privacy and utility: They can offer better utility than "trivially" private architectures which use no edge information (e.g., vanilla MLPs) and better resilience against existing link-stealing attacks than vanilla GCNs which use the full edge structure. LPGNet also offers consistently better privacy-utility tradeoffs than DpGCN, which is the state-of-the-art mechanism for retrofitting differential privacy into conventional GCNs, in most of our evaluated datasets.

CCS CONCEPTS

• **Information systems** → **Clustering and classification**; • **Security and privacy** → **Data anonymization and sanitization**; *Privacy-preserving protocols*; • **Computing methodologies** → **Supervised learning by classification**; *Neural networks.*

KEYWORDS

Graph neural networks; Differential privacy; Node classification; Link-stealing attacks; Machine Learning on graphs

CCS '22, November 7–11, 2022, Los Angeles, CA, USA

ACM ISBN 978-1-4503-9450-5/22/11.
https://doi.org/10.1145/3548606.3560705

ACM Reference Format:
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 (CCS '22), November 7–11, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3548606.3560705

1 INTRODUCTION

Graph neural networks (GNN) learn node representations from complex graphs similar to how convolutional neural networks do from grid-like images. One of the prominent uses of GNNs is to classify the graph nodes based on their node features [28, 53, 62]. They are applied to graphs arising in social networks [17, 54], computer vision [49, 51], natural language processing [2], and traffic prediction [64]. GNNs are deployed into large-scale recommender systems at Pinterest [63] and Google Maps [31]. Graph convolutional networks (GCNs) are one of the most successful type of GNNs for node classification and offer close to state-of-the-art performance [28]. Hence, in this work, we will focus on GCNs for concreteness.

Although node classification can be done purely with the knowledge of node features, the graph edge structure is known to help achieve better accuracy in the classification task. This is why GCNs are popularly used in such tasks. GCNs directly encode, as one of its hidden layers, the adjacency matrix representation of the edges of the given graph. Therefore, they can internally compute features of a node from that of its neighbors. Compared to using vanilla neural networks, such as deep multilayer perceptrons (MLP), they can provide better accuracy in node classification tasks.

In many applications, however, graph edges correspond to sensitive social or financial relationships between people represented as nodes in the graph. Since these are directly used in training GCNs, the privacy risk of leaking which edges are present in the graph is a serious concern. In fact, attacks which can decipher edges present in the original graph used for training given black-box access to the trained GCNs have recently been shown [22, 59]. For instance, the LinkTeller attack is reported to infer edges from GCNs with high precision without needing any background knowledge.

To defend against such attacks, a principled approach is to use differential privacy (DP) in the GCN training process. The first algorithm to train GCNs with DP guarantees for graph edges, called DpGCN, was proposed recently [59]. DpGCN, much like other DP algorithms, adds noise to hide whether each individual private data

element, i.e., an edge, is present. The noise added is based on a privacy budget parameter called ϵ which determines its privacy-utility tradeoffs. The main issue with DPGCN is that it achieves poor privacy-utility tradeoff. DPGCN models trained with even moderate privacy budgets ($\epsilon \in [1, 5]$) perform worse than a standard MLP by up to 44% in classification accuracy on standard datasets. This implies that MLPs, which only use node features and not the graph edges at all, fare better than DPGCN. At privacy budgets ($\epsilon > 8$) DPGCN may perform better than an MLP, but existing attacks achieve similar attack performance on the DPGCN models as they do on the non-private GCNs [59]. This indicates that DPGCN offers little privacy at such ϵ. In short, DPGCN offers very few *sweet spots* where one can reap the utility benefits of using edges for training while offering some privacy advantage over GCNs.

Our Approach. In this work, we present a new neural network architecture called **L**ink **P**rivate **G**raph **Net**works or LPGNET that offers better privacy-utility tradeoffs. *LPGNET often hits the sweet spots and offers significantly better attack resilience than DPGCN while offering similar or better utility.* Our key insight is to move away from the conventional GNNs/GCNs where the raw graph edges (adjacency matrix) are an inseparable part of the neural network architecture. Conventional methods of providing DP such as those used in DPGCN drastically change the adjacency matrix and after adding noise can significantly distort the propagation structure inside a GCN. Instead, we propose to separate the graph (edge) structure from the neural network architecture and only query the graph structure when needed. To achieve this, we design a novel architecture using only MLPs to model both the node feature information and some carefully chosen graph structural information. The structure information, which is provided as features to the MLPs, helps neighboring nodes have more similar representation in the feature space used by MLPs.

What kind of structural information should one use? Observe that GCN-based classification works by exploiting the phenomenon of "homophily" which says that neighboring nodes often have similar features, therefore a node's cluster is likely to be the same as that of the majority of its neighbors [35]. This phenomenon has been extensively observed in real-world graphs. In fact, GCNs work well because they use the raw edge structure to internally aggregate cluster labels from a node's neighbors. On the other hand, MLPs which do not use the edge structure at all can easily classify nodes into a cluster different from that of the majority of its neighbors, which does not adhere to the principle behind homophily.

Our insight is thus to compute a special representation on the graph structure called a *cluster degree vector*. For each node, the cluster degree vector query asks: "How many neighbors of the node are present in each cluster". We observe that if the level of homophily in a graph is high, then nodes in the same cluster will have cluster degree vectors of high cosine similarity. These vectors capture information about the level of homophily in the graph, and at the same time, use only coarse-grained node degree counts rather than using individual edges. Based on our above observations, our novel LPGNET architecture stacks layers of MLPs trained on node feature embeddings and splices noisy (differentially private) cluster degree vectors between successive stacked layers. This improves the classification accuracy iteratively after each stacked layer because nodes in the same cluster have similar cluster degree vector representation. Privacy sensitive edges are only queried while computing these cluster degree vectors.

Evaluation. We evaluate both the transductive setting, where the inference and training are performed using different nodes of the same graph, and the inductive setting, where the inference graph is different (as in transfer learning). We use 6 standard benchmark datasets. In the transductive setting, we find that LPGNET has *consistently better* privacy-utility tradeoffs compared to the state-of-the-art defense, namely DPGCN. LPGNET outperforms DPGCN in classification accuracy by up to 2.6× in $\epsilon \in [1, 2]$ and remains superior for all $\epsilon \in (2, 7]$ and across *all* datasets evaluated. DPGCN offers significantly worse utility compared to a vanilla MLP (which is trivially private) for all $\epsilon \in [1, 7]$, whereas LPGNET performs better than MLP at all $\epsilon \geq 2$ and close to it for $1 \leq \epsilon < 2$ on all datasets. We also evaluate two state-of-the-art link stealing attacks, LPA [22] and LINKTELLER [59], on our proposed defense. [1] LPGNET has better attack resilience than DPGCN whenever both of them have a better utility than an MLP. The attack performance, measured by the Area Under the Receiver Operating Curve (AUC), is at most 11% higher on LPGNET compared to MLP in absolute difference across all datasets and both attacks. In contrast, on DPGCN the best attack's AUC could go up to 22% higher than MLP. Further, LINKTELLER regularly achieves an AUC > 0.95 on DPGCN which implies that, in such cases, DPGCN offers no protection to a GNN that leaks edges to LINKTELLER. In the inductive setting, LPGNET also exhibits better utility than DPGCN for a majority of evaluated configurations. The attack AUC on LPGNET is always lower than DPGCN at similar utility and can be up to 34% lower than DPGCN.

Finally, we discuss how to interpret our results and conclude what attacks LPGNET (or DP in general) protects against. Our discussion contains new insights and pointers to future work which may not be obvious from the prior attacks on graph learning.

Contributions. We propose LPGNET, a new stacked neural network architecture for learning over graphs. It offers differential privacy guarantees for graph edges used during training. LPGNET is designed to better retain the signal of homophily present in training graphs without directly using fine-grained edge information. We compare LPGNET to state-of-the-art DP solution (DPGCN). We show that LPGNET has better resilience against existing link stealing attacks than DPGCN models with the same level of utility.

Availability

LPGNET is publicly available on GitHub. [2] We refer to the Appendices several times throughout the text. They can be found in the full version of this paper [29].

2 PROBLEM

GNNs learn low-dimensional representations for nodes (node embeddings) that are used to classify them into meaningful categories. However, to achieve that, GNNs may have to be trained on private data that is stored across multiple entities. We provide the overview of our application setup and the problem in Figure 1.

[1] The LPA paper uses LSA to denote their attack, we call it LPA.
[2] https://github.com/ashgeek/lpgnet-prototype

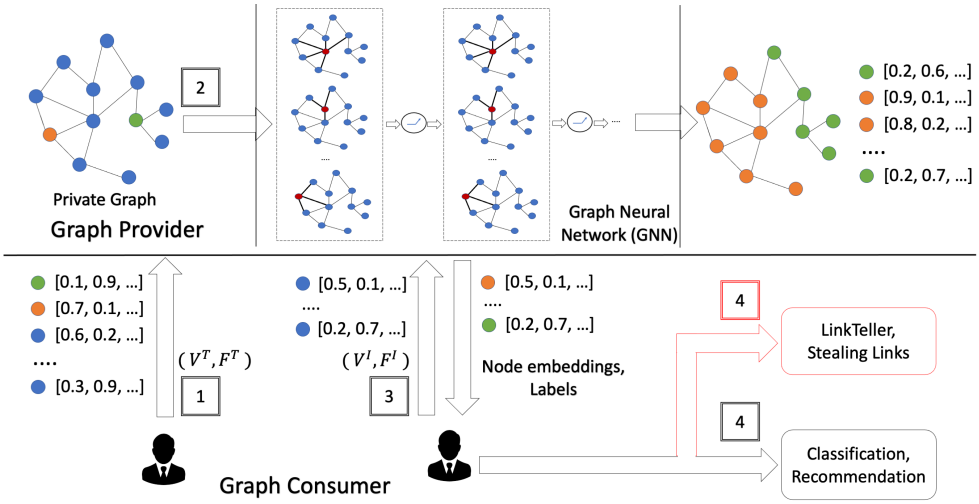

Figure 1: The private graph of users is stored by the graph provider service which is queried by a graph consumer service that has user attributes. In step 1, the graph consumer service sends a set of user features and their labels to the graph provider service. In step 2, the graph provider uses the features and its private graph to train a graph neural network and releases a black-box query access API. The graph consumer queries the API with a new set of node features to infer their node embeddings and their labels. In step 4, the node embeddings may be used for downstream applications such as node classification and recommendations. However, the graph consumer may also attack the GNN using the black-box API to infer the private edges.

2.1 Application Setup & Goals

We focus on the problem of semi-supervised node classification over a graph in which the edges are privacy-sensitive. To illustrate this, consider a trusted *graph provider* (Facebook) which hosts a real-world social networking graph. Its nodes are users and the edges denote their private social links. Consider an untrusted *graph consumer*, for instance a music streaming service (Spotify), that holds features of the users such as their music preferences, but not their social links. It may want to use the graph structure along with the user features to improve its recommendations. The graph structure is important since it encodes the principle of "homophily", i.e., the neighboring nodes on the graph tend to have similar interests.

The aforementioned example is studied as an instance of a more general problem called the *node classification* problem. The goal is to classify nodes that are similar to each other into clusters with distinct *labels*, based on their features and the edge structure. In the semi-supervised setup, the cluster labels of a small set of nodes are known and the task is to predict the labels of the remaining nodes. Specifically, a graph $\mathcal{G}^T : (\mathcal{V}^T, \mathcal{E}^T)$ is available to the graph provider and the features $\mathbf{F}^T$ along with the labels $\mathbf{Y}$ for a small set of nodes in $\mathcal{V}^T$ are available to the graph consumer.

In this well-studied application setup, the graph consumer gives the features $\mathbf{F}^T$ and labels $\mathbf{Y}$ to the graph provider. The provider then uses them along with the graph $\mathcal{G}^T$ to train an ML model and allows the consumer to query the model hosted on its servers. The consumer can then query the ML model with a set of *inference nodes* $\mathcal{V}^I$ along with their features $\mathbf{F}^I$ to get their low-dimensional embeddings that are output by the ML model [22, 59].

The learnt model can be queried for labels of inference nodes, which were not part of the training set. There are two settings in the literature for the classification task. In the *transductive* setting, the training and inference are performed on the same graph. The goal is to learn a classifier using the entire graph structure and a few labeled nodes to predict labels of other nodes in it. In the *inductive* setting, the graphs used for training and inference are different. The classifier is evaluated on how transferable it is to unseen graphs (or different versions of a dynamically changing graph).

Background: Graph neural networks. Graph neural networks (GNNs) achieve state-of-the-art performance for the node classification problem. Several GNN architectures have been proposed recently, however, we describe one widely-used architecture called the graph convolutional neural network (GCN) for concreteness [28]. It takes a matrix as input with each row corresponding to the features (vector) of a user or node in $\mathcal{V}$. The logits obtained from the last layer of the GCN are the node embeddings which can be converted to probability scores per label using a softmax layer. A GCN directly splices the graph's edge structure in its internal architecture itself. In every hidden layer, for each node in the graph the GCN aggregates the features obtained from all its neighbors. Therefore, when k hidden layers are stacked, the features are obtained from all k-hop neighbors for aggregation. Formally, every hidden layer $\mathbf{H}^k$ in a GCN is represented as follows:

$$\mathbf{H}^k = \sigma(\tilde{\mathbf{A}} \cdot \mathbf{H}^{k-1} \cdot \mathbf{W})$$

$\tilde{\mathbf{A}}$ is a normalized adjacency matrix, $\mathbf{H}^0$ is the input node features, $\mathbf{W}$ is the weight parameters and σ is a non-linear activation function.

Privacy Goals. Revealing the graph edges to the untrusted data consumer is a serious privacy issue. Many prominent social networking companies protect against querying the direct social links of the user. Our goal is to ensure that the untrusted data consumer does not learn about the presence or absence of edges in the data

provider's graph by querying the ML model that uses the graph for training or inference. We assume that the untrusted consumer only has black-box access to the trained model. Even in this restricted setup, recent works [22, 59] have shown that it is possible to identify if an edge was used in training with high (nearing 100%) precision in some cases. We aim to train GNNs that are less likely to reveal that they are using an edge for training or inference using the differential privacy (DP) framework [16]. By definition, such GNNs are *resilient* to link-stealing attacks that infer whether specific edges were used by the GNN during training or inference.

A differentially private query will ensure that the adversary can get only a bounded amount of additional information about the input dataset after seeing the query outputs.

Definition 2.1 (Differential Privacy)**.** Consider any two datasets, $\mathcal{D}_1, \mathcal{D}_2 \in \mathcal{D}$, such that ($|\mathcal{D}_1 - \mathcal{D}_2| \leq 1$). A randomized query $\mathbf{M} : \mathcal{D} \rightarrow \mathcal{S}$ satisfies ϵ-DP if, for all such $\mathcal{D}_1, \mathcal{D}_2 \in \mathcal{D}$ and for all $\mathcal{S}_0 \subset \mathbf{M}(\mathcal{D})$,

$$Pr(\mathbf{M}(\mathcal{D}_1) \in \mathcal{S}_0) \leq e^{\epsilon} \cdot Pr(\mathbf{M}(\mathcal{D}_2) \in \mathcal{S}_0)$$

For graphs, there are 2 notions of DP: edge-DP [21] and node-DP [27]. Edge-DP is used to protect the existence of any individual edge in the graph. Node-DP is used to protect the existence of any node (and its edges). DPGCN is designed with edge-DP as its objective. In our work, we also use the edge-DP framework since our goal is to protect individual edges from being leaked by a GNN. Achieving node-DP is promising future work and usually has worse utility trade-offs than edge-DP in other setups.

We consider undirected graphs with unweighted edges in this work. If $\mathcal{V}$ is given as an ordered set of nodes then every undirected graph $\mathcal{G}$ that can be constructed with $\mathcal{V}$ is captured by an array of 1s and 0s, i.e., $\mathcal{D}_{\mathcal{G}} = \{I(v_i, v_j) | i, j \in \{1, 2, \ldots, |\mathcal{V}|\} \text{ and } i < j\}$. Here, $I(v_i, v_j) = 1$ if the nodes v_i and v_j have an edge in $\mathcal{G}$ otherwise 0. Therefore, to define an edge-DP mechanism, we just view $\mathcal{D}$ in definition 2.1 as a set of all possible $\mathcal{D}_{\mathcal{G}}$ and $\mathcal{D}_1, \mathcal{D}_2$ as two datasets (graphs) with a difference of one edge (addition or removal).

2.2 Existing Solution

Recently, Wu et al. [59] have proposed an edge-DP algorithm to learn a GCN, called DPGCN. In order to achieve edge-DP, DPGCN proposes to use the Laplace mechanism on the adjacency matrix before training the GCN. Here, we define the Laplace mechanism:

Definition 2.2 (Laplace Mechanism)**.** Let the sensitivity of the query $f : \mathcal{D} \rightarrow \mathcal{S}$ be $\Delta(\mathbf{f}) = \max |\mathbf{f}(\mathcal{D}_1) - \mathbf{f}(\mathcal{D}_2)|$ for all $\mathcal{D}_1, \mathcal{D}_2 \in \mathcal{D}$ such that $|\mathcal{D}_1 - \mathcal{D}_2| \leq 1$. The mechanism $\mathbf{M} : \mathcal{D} \rightarrow \mathcal{S}$ defined as follows satisfies ϵ-DP:

$$\mathbf{M}(\mathcal{D}_1) = \mathbf{f}(\mathcal{D}_1) + \text{Lap}(0, \frac{\Delta(\mathbf{f})}{\epsilon})$$

Once the adjacency matrix is differentially private the rest of the training process is also differentially private as given by the *post-processing* property of DP mechanisms [16]. Formally,

Definition 2.3 (Post-processing property)**.** If a mechanism $\mathbf{M}$ satisfies ϵ-DP then any function $\mathbf{g} \circ \mathbf{M}$ also satisfies ϵ-DP.

Algorithm 1 summarizes the DPGCN solution. For undirected graphs, the adjacency matrix can be represented by an upper triangular matrix. Therefore, if we add or remove an edge from the graph then only one entry in the adjacency matrix will change. So, adding noise sampled from $\text{Lap}(0, \frac{1}{\epsilon})$ to each entry in adjacency matrix results in a differentially private one. However, now the adjacency matrix becomes a weighted matrix with a non-zero weight associated with almost every entry (edge). Hence, the noised adjacency matrix represents a very dense and weighted graph. In order to maintain the original graph statistics such as the exact number of edges ($E = |\mathcal{E}|$) and density, they propose to take the top-E entries in the DP-adjacency matrix to be the actual edges (replace the top-E with 1 and rest 0). A small privacy budget $\epsilon_r < \epsilon$ is used to compute the number of edges ($\tilde{E} = E + \text{Lap}(0, \frac{1}{\epsilon_r})$). The remaining privacy budget $\epsilon - \epsilon_r$ is used to noise the adjacency matrix.

Algorithm 1: An outline of DPGCN [59]. It adds Laplace noise to all entries of the adjacency matrix and selects the top-$\tilde{E}$ values to maintain the original density of the graph.

Input : Graph $\mathcal{G} : (\mathcal{V}, \mathcal{E})$ as adjacency matrix $\mathbf{A}$, Total privacy budget: ϵ, Privacy budget for estimating the number of edges: $\epsilon_r = 0.01$ (default).
Output : $\hat{\mathbf{A}}$

$E = |\mathcal{E}|$;
$\tilde{E} = E + \text{Lap}(0, \frac{1}{\epsilon_r})$;
$\tilde{E} = \lfloor \tilde{E} \rfloor$;
$\mathbf{A}_{tr} = \text{UppTriMatrix}(\mathbf{A}, 1)$;
`// Upper triangular matrix on` 1^{st} `superdiagonal`
for *entry in* $\mathbf{A}_{tr}$ **do**
  $\mathbf{A}_{tr}[entry] += \text{Lap}(0, \frac{1}{\epsilon - \epsilon_r})$;
end
`// neglecting entries of the lower triangle`
$top\_indices = arg\,max(\mathbf{A}_{tr}, \tilde{E})$;
`// get the indices of top` $\tilde{E}$ `values`
$\mathbf{A}_{tr}[top\_indices] = 1, \mathbf{A}_{tr}[\neg top\_indices] = 0$;
$\hat{\mathbf{A}} = \mathbf{A}_{tr} + \mathbf{A}_{tr}^T$;
return $\hat{\mathbf{A}}$

2.3 Attacks & Utility Tradeoffs

DP mechanisms trade off utility for privacy by adding noise. In DP, ϵ measures the level of privacy—lower ϵ means higher privacy. The utility is expected to decrease with decreasing ϵ. There is no consensus on how to choose ϵ in practice, however, ϵ values ranging from 0.1 to 10 have been used consistently to evaluate newly proposed mechanisms [1, 6, 33, 59, 61]. For our problem, existing approaches offer a glaringly poor utility-privacy tradeoff.

To illustrate the poor tradeoff, we show the node classification performance of DPGCN on a commonly used dataset called Cora at $\epsilon \in [1, 10]$, following the evaluation in prior work [59], in Figure 2. Strikingly, the performance is worse than a two-layer fully connected feed-forward neural network (MLP) for $\epsilon < 7$. An MLP trivially offers DP since it does not use the graph edges at all. This means that if one desires $\epsilon < 7$, then using an MLP would be better than models produced by DPGCN. On the other hand, when the $\epsilon \geq 8$, DPGCN performs better than the MLP and closer to the GCN. However, at that ϵ, LinkTeller can predict the edges in the

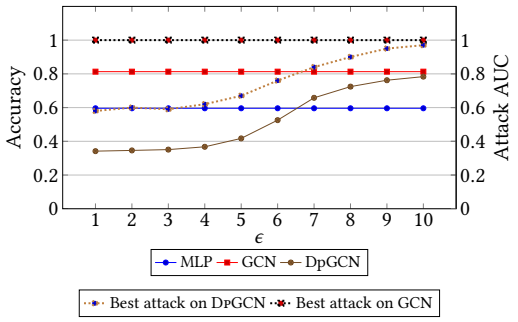

Figure 2: The privacy-utility tradeoffs offered by DpGCN on Cora dataset. It does not perform better than an MLP until $\epsilon = 7$ making it unusable at such ϵ. At higher $\epsilon \geq 9$ the attack performance on DpGCN is as high as its performance on non-DP GCN making it unusable there as well.

original graph with close to 100% precision and an AUC score ≥ 0.9. The attack performance on DpGCN is very close to its performance on the non-DP GCN. This shows that at $\epsilon \geq 8$ the model offers as much privacy as a non-private GCN. We observe similar trends for other datasets as well, confirming the findings in [59].

There are few ϵ values for which the attack accuracy is weaker on DpGCN compared to a non-DP GCN and utility is better than that of MLPs. For example, see $\epsilon \in [7, 8)$ for the Cora dataset in Figure 2. We call such values sweet spots. Intuitively, when the number of sweet spots are high, the private algorithm is more useful as a defense against existing attacks since its utility is better than MLP's which does not use edge structure. Specifically, a defense is objectively better than another defense in a particular setting if the attack performance on that defense is worse, while the utility is better than the other defense. Figure 2 shows how DpGCN fares worse than MLP for $\epsilon < 7$, offering no sweet spots in that area. Stated succinctly, our main challenge and central contribution is to show a new way of designing neural network models that often hit sweet spots between privacy and utility. The performance desired should be better than MLP, which does not use edges, and DpGCN which does. Further, it should offer lower (better) ϵ values, and be as resistant as an MLP to state-of-the-art attacks (LinkTeller).

3 APPROACH

Let us see why the DpGCN approach offers a poor privacy-utility tradeoff, which motivates our new approach.

Top-down approach does not work. DpGCN takes a top-down approach wherein it retrofits DP to an existing GNN architecture trained for node classification. All GNN architectures use the adjacency matrix in some form or another. Adding noise to the adjacency matrix and using it in the GNN provides DP, but notice

Table 1: Percentage of Noisy edges sampled by DpGCN for the Cora dataset with varying $\epsilon \in [1, 10]$.

ϵ	1	2	3	4	5	6	7	8	9	10
Noise	100	99	98	93	84	66	42	25	15	9

that the adjacency matrices have very high sensitivity. Even when a small noise sampled from $Lap(0, \frac{1}{\epsilon})$ is added to each entry of the matrix, it can create $O(|V|^2)$ weighted edges which may not exist in the original graph. We observe that when the DpGCN selects top-E edges among them, it cannot differentiate between the original and "noisy" edges at ϵ such as say $\epsilon = 2$. Thus, it selects a lot of noisy edges and the number of such edges gradually increases as ϵ reduces. To illustrate, 99% of the selected top-E edges are noisy at $\epsilon = 2$ for a standard Cora dataset, as shown in the Table 1. This explains why the DpGCN has poor utility-privacy tradeoffs. At moderately low ϵ values, the graph structure is significantly different from the original which leads to loss of utility. At high ϵ values, the noise added is small and most of the selected edges are original graph edges, but then at this level, the link-stealing attacks work as effectively on DpGCN as on non-private GCNs.

Our approach. We propose LPGNet, an ML model architecture that can give good utility while avoiding the direct use of the edge structure. LPGNet starts with an MLP trained only over node attributes since it is the basic component of most of the GNN architectures. Next, and unlike typical GNNs, LPGNet computes a *cluster degree vector* for each node that captures coarse-grained graph structure information. To compute cluster degree vector, nodes are first partitioned into clusters obtained from the labels predicted by the previous MLP stack layer. For every node LPGNet computes the number of neighbors it has in each of those clusters. The cluster degree vector for each node is thus an array of those counts, indexed by the cluster label. The cluster degree vector is then used with node attributes to train the next MLP stack layer. The full architectural details of LPGNet are given in Section 4.

Let us see why our approach works. The idea behind GNNs is that a node tends to have the same label as the majority of its neighbors, a property called homophily which is observed in many real-world networks. But, training an MLP without the edge structure can often result in the following scenarios which do not adhere to homophily: 1) A node receives a wrong label which is different from the correct label received by a majority of the node's neighbors, 2) A node receives a wrong label and its neighbors receive correct labels but there is no majority label, and 3) A node receives wrong label and its neighbors also receive wrong labels. GNNs suppress these scenarios as the model combines the features of a node's neighbors using the adjacency matrix. This local aggregation pushes two nodes that have many common neighbors towards having similar aggregated features, and eventually, the same cluster labels at each layer.

In LPGNet, we indirectly achieve the same goal without using the raw or noisy edge structure directly. We can empirically observe how homophily structure of the graph is preserved in our architecture compared to the alternatives. Specifically, let the fraction of a node's neighbors that reside in the same cluster as itself be its homophily score and $AvgHomophily(i)$ be the average of the homophily scores of all nodes in cluster i. If we plot $AvgHomophily(\cdot)$ for all clusters, we obtain what we call a *homophily plot*. Figure 3 shows homophily plots for the Cora dataset. The leftmost plot is for the ground truth clusters in the original graph and the phenomenon of homophily is clearly evident. For instance, nodes of cluster 0 have about 80% of their neighbors in the same cluster, nodes in cluster 2 have about 90% of nodes in the same cluster, and so on.

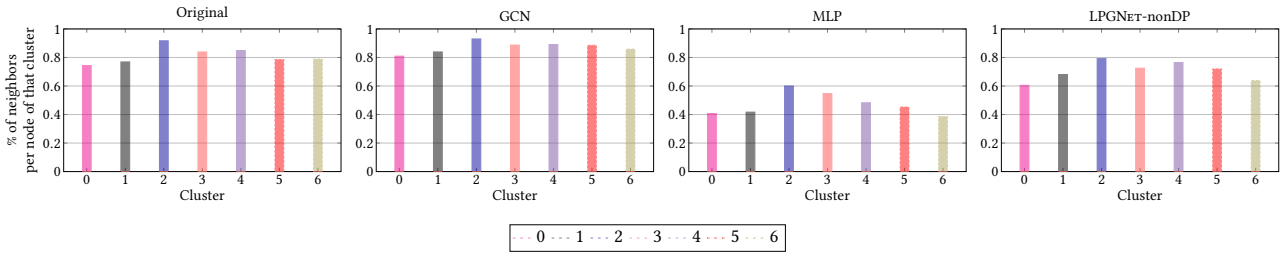

Figure 3: Homophily plots for the Cora dataset. The leftmost plot is for the original graph, the next two for a non-private GCN and MLP respectively, and the rightmost for our LPGNet without any noise added.

If we take the clusters produced by non-private GCN (the second plot in Figure 3), the $AvgHomophily(\cdot)$ seems to be very similar to the ones produced with the ground truth clusters. Notice that with clusters obtained by the MLP, however, the $AvgHomophily(\cdot)$ is significantly lesser than ground truth. This shows how MLP preserves much weaker homophily in its output than in the ground truth. Our key idea in LPGNet is to sharpen the homophily signal, if it exists in the graph used for training, in each stack layer. Observe that homophily implies that two nodes that belong to the same cluster will have similar cluster degree vectors[3] as a majority of their neighbors will also belong to the same cluster. By using cluster degree vectors along with the node features to train the next MLP stack layer, our architecture is giving similar cluster degree vectors to nodes in the same cluster, thus increasing the signal-to-noise ratio [34]. The last homophily plot shows that the LPGNet architecture preserves the homophily characteristics closer to how GCNs do, at least when no noise is added during training.

Achieving edge-DP is simple in LPGNet. We add noise to the cluster degree vectors as they are the only computation being made on the graph. If we change one edge in the graph then the degree vectors of the two involved nodes will have changed by 1 each. Therefore, a noise sampled from $Lap(0, \frac{2}{\epsilon})$ can be added to each count of degree vectors of all nodes. The noise added to degree vectors will have much less impact on them unlike the noise added to the adjacency matrix as they are more coarse-grained than the exact edge information. Specifically, each array count will change by more than $\frac{2}{\epsilon}$ only about 25% of the time and $\frac{4}{\epsilon}$ only 5% of the time. [4] At $\epsilon = 1$, the most significant array count that may represent the node's actual cluster can be much higher than the noise added depending on the node's degree. In contrast, the same amount of noise added to an adjacency matrix completely changes the graph since the actual values (1s and 0s) fall within the standard deviation of the noise itself.

4 ARCHITECTURAL DETAILS OF LPGNET

LPGNet represents and implements a chain of MLPs that are sequentially trained (one after another independently) in an iterative fashion while taking the help of degree vectors in each iteration. Figure 4 captures LPGNet's architecture and the training process.

Multi-MLP architecture. LPGNet first trains an MLP on only the node features to obtain the initial set of clusters. Specifically, the input is the feature matrix ($\mathbf{F}^T$ of size $N \times F$) and the output is a node embedding ($\mathbf{L}$ of size $N \times C$), where $N = |\mathcal{V}^T|$, F is the input feature dimension and C is the number of labels (clusters). Node embeddings are the logit scores output by the MLP, therefore, we can apply the softmax function over the node embeddings to get the labels for each node. Using these labels LPGNet obtains the first clusters without any edge information. LPGNet then begins the training of additional MLPs iteratively.

In each iteration (stack layer) i, LPGNet trains an MLP based on the degree vectors computed from the clusters obtained in the previous iteration. The degree vectors can be represented by an $N \times C$ matrix ($\mathbf{X}$) where each row is an array of counts for each node. LPGNet concatenates the degree vector matrix obtained in the previous iteration $\mathbf{X}_{i-1}$ to the node embedding matrix from the previous iteration $\mathbf{L}_{i-1}$, thus, creating $\mathbf{F}_i = \mathbf{L}_{i-1} \frown \mathbf{X}_{i-1}$ an $N \times 2C$ matrix that comprises of both node features and the edge information. [5]

Entire history for training. We observe that LPGNet's performance improves by using the features obtained from all the previous iterations for the current iteration. Specifically, the features in the iteration $i+1$, are the concatenation of $\mathbf{X}_i$, $\mathbf{L}_i$, and $\mathbf{F}_i$ where $\mathbf{F}_i$ itself is computed recursively until $i = 1$, i.e., $\mathbf{F}_{i+1} = \mathbf{F}_i \frown (\mathbf{L}_i \frown \mathbf{X}_i)$.

Transductive vs inductive setting. In the transductive setting, LPGNet is trained and tested on the same graph $\mathcal{G}^T : (\mathcal{V}^T, \mathcal{E}^T)$. The features of all nodes are given to the graph provider to train LPGNet. During inference time, a set of nodes $\mathcal{V}^I$ are given to the provider. Note that the provider already has their features $\mathbf{F}^I$. LPGNet outputs the logits that are produced by the last MLP in the training. All the intermediate logits are hidden from the graph consumer. In the inductive setting, LPGNet is trained and tested on different graphs. The training procedure is similar to the transductive setting. For inference, the nodes $\mathcal{V}^I$ and their features $\mathbf{F}^I$ from the new graph are provided to the graph provider. It is assumed that provider already has the new graph $\mathcal{G}^I : (\mathcal{V}^I, \mathcal{E}^I)$ which is usually the case with evolving graphs or when the provider has graphs of disjoint set of users from sometimes different countries [46]. If the training and inference graphs are different (call it the *inductive-different* setting) then we use only the graph $\mathcal{G}^I$ to compute cluster degree vectors for LPGNet during inference time. In contrast, if the graph is evolving (call it the *inductive-evolving* setting) where the

[3] Specifically, we mean cosine similarity here, which ignores the scaling factor due to differing degrees of the two nodes being compared.

[4] This can be computed using the Hoeffding Inequality on the Laplace random variables corresponding to the noise.

[5] $\frown$ denotes concatenation.

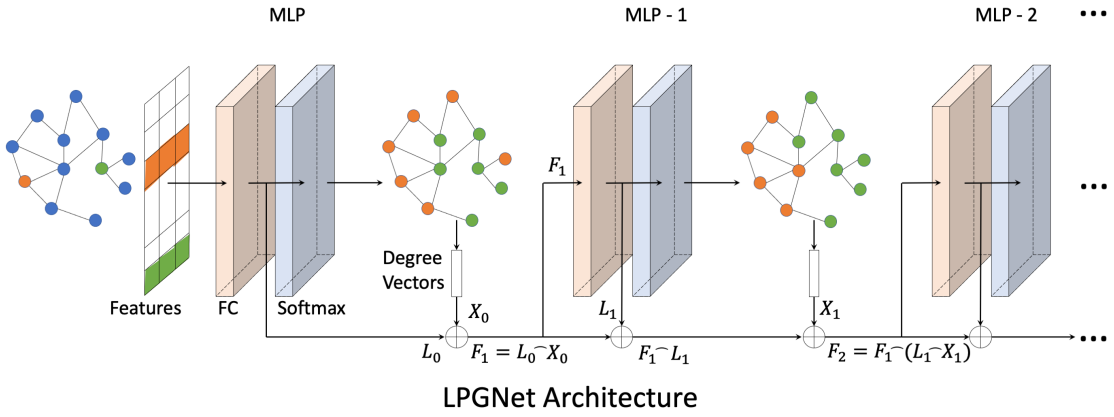

Figure 4: LPGNet trains MLPs iteratively with the node features and additional graph features encoded in the form of degree vectors. The features provided to every MLP is a concatenation of the features obtained from from all the previous MLPs.

edges used for training are a subset of the edges used for inference then the entire graph $\mathcal{G}^I$ (that includes $\mathcal{G}^T$) is used to compute the cluster degree vectors. To summarize the inference procedure in the inductive setting, for each additional MLP, LPGNet computes the cluster degree vectors on $\mathcal{G}^I$ and uses them along with the attributes of $\mathcal{V}^I$ to perform the forward pass on the MLP. Finally, the last MLP's logits are revealed to the graph consumer.

Algorithms 2 and 3 summarize the training and inference algorithms for LPGNet. During the training phase, the best intermediate MLPs are chosen based on the validation loss on the validation dataset whenever it is available. In the transductive setting, the nodes used for validation are from the same graph as the nodes used for training. Therefore, the cluster degree vectors are computed for all nodes once before training every additional MLP in LPGNet and are reused for validation. In the inductive setting, the nodes used for validation may come from a different graph or from the unseen part of an evolving graph. So, we compute the cluster degree vectors again for validation (Lines 17-26 in Algorithm 2).

4.1 LPGNet's Privacy Details

Differential privacy. To satisfy edge-DP, LPGNet adds noise sampled from $\text{Lap}(0, \frac{2}{\epsilon})$ to all counts in the cluster degree vectors whenever they are computed from the graph (Lines 6, 20 in Algorithm 2 and Line 7 in Algorithm 3). Without DP, LPGNet can train many additional MLPs, but with DP LPGNet can train only a few since the privacy budget adds up for every such MLP. For instance, if $\epsilon = 2$ for computing the cluster degree vectors once then after 5 additional MLPs the resulting ϵ is 10.

In our implementation, we fix a privacy budget that is consumed during the training, validation and inference phases. We split that budget between all possible degree vector queries across the three phases. Whenever possible, we save the privacy budget by reusing cluster degree vectors during validation and inference phases (Lines 8, 11 in Algorithms 2 and 3 respectively). LPGNet preserves ϵ edge-DP based on the post-processing property since the all computations build on the DP cluster degree vectors. Concretely, we prove the following theorems to show that LPGNet provides ϵ edge-DP in both the transductive and inductive settings.

Theorem 1. *LPGNet satisfies ϵ edge-DP for the transductive setting.*

Theorem 2. *LPGNet satisfies ϵ edge-DP for the inductive setting.*

We provide the details of how LPGNet uses its privacy budget and the proofs for the aforementioned theorems in Appendix A.

5 EMPIRICAL EVALUATION

Our primary goals are three-fold. First, we want to evaluate how well LPGNet will perform in the node-classification task as compared to MLP and GCN. Second, we want to measure how much utility is traded off when LPGNet is trained with a DP guarantee and check if LPGNet performs better than MLP and DPGCN. Third, we want to compare the privacy-utility tradeoffs offered by LPGNet and DPGCN when the privacy is measured by how well the 2 state-of-the-art attacks— LinkTeller [59] and LPA [22]—work.

5.1 Methodology

Datasets. We use 6 standard datasets for the node classification task used in prior work [28, 46, 59]. In the transductive setting, we use Facebook, Cora, Citeseer, and PubMed datasets. Facebook is a social network and the rest are citation networks. In the inductive setting, we use Twitch and Flickr datasets and both are social networks. Twitch is a collection of disjoint graphs, one from each country. Across all graphs the node features have the same dimension and the semantic meaning. The task is to classify whether a Twitch node (streamer) uses explicit language. We train GNNs on the graph corresponding to Spain and use other graphs for testing as done previously [59]. Flickr dataset contains an evolving graph. We train GNNs on one part of the graph and test it on the evolved

Algorithm 2: The training algorithm for LPGNet. In the transductive setting, the validation graph and the training graph are the same. For inductive, they can be different hence we recompute the degree vectors on validation graph.

Input : Train graph $\mathcal{G}^T : (\mathcal{V}^T, \mathcal{E}^T)$, Train Features and labels: $(\mathbf{F}^T, \mathcal{Y}^T)$, Validation graph $\mathcal{G}^V : (\mathcal{V}^V, \mathcal{E}^V)$ (for inductive), Validation Features and labels: $(\mathbf{F}^V, \mathcal{Y}^V)$, Privacy budget: ϵ, LPGNet size (# of additional MLPs): $nl >= 1$, Hyperparameters: learning rate (lr), dropout rate (dr), hidden layer size (hid_s), # hidden layers (hid_n), # of epochs (e)

Output : Trained LPGNet $\mathbf{M}$

$\mathbf{M}[0] = \text{trainMLP}(\mathbf{F}^T, \mathcal{Y}^T, \mathbf{F}^V, \mathcal{Y}^V, hid_n, hid_s, lr, dr)$;
// select the best model across e epochs using Adam optimizer and cross-entropy loss.
$i = 0, \mathbf{F}_0 = \mathbf{F}^T, {\mathbf{F}_0}^V = \mathbf{F}^V$;
while $i < nl$ **do**
  $\mathbf{L}_i = \mathbf{M}[i].\text{forward}(\mathbf{F}_i)$;
  // get logits from one forward pass
  $labels = \text{softmax}(\mathbf{L}_i)$;
  $\mathbf{X}_i = \text{findDegreeVec}(\mathcal{G}^T, labels, \frac{\epsilon}{nl})$;
  if *transductive setting* **then**
    Store to disk $\mathbf{X}_i$;
  end
  if *i is 0* **then**
    $\mathbf{F}_{i+1} = \mathbf{L}_i \frown \mathbf{X}_i$;
    // $\frown$ denotes concatenation
  end
  else
    $\mathbf{F}_{i+1} = \mathbf{F}_i \frown (\mathbf{L}_i \frown \mathbf{X}_i)$;
  end
  // concatenate logits and degree vectors
  ${\mathbf{F}_{i+1}}^V = \mathbf{F}_{i+1}$ // for transductive
  if *validation on different graph* **then**
    ${\mathbf{L}_i}^V = \mathbf{M}[i].\text{forward}(\mathbf{F}_i^V)$;
    $validationlabels = \text{softmax}({\mathbf{L}_i}^V)$;
    ${\mathbf{X}_i}^V = \text{findDegreeVec}(\mathcal{G}^V, validationlabels, \frac{\epsilon}{nl})$;
    if *i is 0* **then**
      ${\mathbf{F}_{i+1}}^V = {\mathbf{L}_i}^V \frown {\mathbf{X}_i}^V$;
    end
    else
      ${\mathbf{F}_{i+1}}^V = {\mathbf{F}_i}^V \frown ({\mathbf{L}_i}^V \frown {\mathbf{X}_i}^V)$;
    end
    // concatenate validation logits and degree vectors
  end
  $\mathbf{M}[i+1] =$ $\text{trainMLP}(\mathbf{F}_{i+1}, \mathcal{Y}^T, {\mathbf{F}_{i+1}}^V, \mathcal{Y}^V, hid_n, hid_s, lr, dr)$;
  $i = i + 1$;
end
return $\mathbf{M} = [\mathbf{M}[0], \ldots, \mathbf{M}[nl]]$

Algorithm 3: The inference algorithm for LPGNet. In the transductive setting, the inference graph is same as the train graph $\mathcal{G}^T$ whereas in the inductive setting it is different. Therefore, in the inductive setting the degree vectors are computed on the new graph. Further, only the embeddings (logits) computed on the last MLP are released.

Input : Inference graph $\mathcal{G}^I : (\mathcal{V}^I, \mathcal{E}^I)$, Inference Features: $\mathbf{F}^I$, Privacy budget: ϵ, Trained model: $\mathbf{M}$.

Output : Node embeddings: $\mathbf{L}$ for $\mathcal{V}^I$

nl =# of additional MLPs in $\mathbf{M}$;
$i = 0, \mathbf{F}_0 = \mathbf{F}^I$;
while $i < nl$ **do**
  $\mathbf{L}_i = \mathbf{M}[i].\text{forward}(\mathbf{F}_i)$;
  // get logits from one forward pass
  $labels = \text{softmax}(\mathbf{L}_i)$;
  if *inductive setting and first time inference* **then**
    $\mathbf{X}_i = \text{findDegreeVec}(\mathcal{G}^I, labels, \frac{\epsilon}{nl})$;
    Store to disk $\mathbf{X}_i$;
    // for future inference
  end
  else
    $\mathbf{X}_i \leftarrow$ Use Stored Degree Vectors;
  end
  if *i is 0* **then**
    $\mathbf{F}_{i+1} = \mathbf{L}_i \frown \mathbf{X}_i$;
  end
  else
    $\mathbf{F}_{i+1} = \mathbf{F}_i \frown (\mathbf{L}_i \frown \mathbf{X}_i)$;
  end
  // concatenate logits and degreevectors
  $i = i + 1$;
end
return $\mathbf{L} = \mathbf{M}[nl].forward(\mathbf{F}_i)$

Algorithm 4: Compute degree vectors.

Input : Graph $\mathcal{G} : (\mathcal{V}, \mathcal{E})$, $labels$, Privacy budget: ϵ

Output : Degree vectors: $\mathbf{X}$

Function findDegreeVec($\mathcal{G}$, $labels$, ϵ)**:**
  $Cl = \text{getClusters}(labels)$;
  // nodes with same label is a cluster
  for *node v in* $\mathcal{G}$ **do**
    for *cluster c in* Cl **do**
      n = # of neighbors(v) in c;
      $\mathbf{X}[v][c] = n + \text{Lap}(0, \frac{2}{\epsilon})$;
      // no need to noise for non-DP
    end
  end
  return $\mathbf{X}$

graph. We provide details of the datasets, their graph statistics and the test-train splits in Appendix B.

Model architectures. We have 3 model types for all our experiments: GCN (or DpGCN), MLP, and LPGNet. For GCN, we adopt the standard architecture as described in Section 2 along with ReLU activation and dropout for regularization in every hidden layer. For MLP, we use a fully-connected neural network with ReLU activation and dropout in every hidden layer. In LPGNet, we use the same MLP architecture for all stacked layers however with different input size for each layer. We use a softmax layer to compute the labels from the logits generated by every model. The number of hidden layers, dropout rate and size of the hidden layers are hyper-parameters which can be tuned. We find the right parameters by performing a grid search over a set of possible values in the non-DP setting which we provide in Appendix B. In the DP setting, we evaluate using the best hyper-parameter values from the non-DP setting. There may be better values for the DP setting but finding them would require using additional privacy budgets and we leave such tuning for future work. We also believe that our conclusions will only be stronger with better parameter tuning for DP models.

Training procedure and metrics. We train our models using cross-entropy loss, Adam's optimizer, and weight decay. We evaluate the node-classification performance using the micro-averaged F1 score. As proposed by the LinkTeller paper, we use a modified F1 score for Twitch datasets which computes the F1 score for the rare class as there is a significant class imbalance in the datasets. We provide more details for our training procedure and metrics in Appendix B.

Existing attacks. LinkTeller is based on the observation that two neighboring nodes will influence each other's predictions more than two non-neighboring nodes. To compute the influence, LinkTeller perturbs the features of a target node and observes the changes in the GNN outputs for every other node. For a k-layer GCN, only the nodes within k hop distance of the target node on the graph are influenced. For a 1-layer GCN the LinkTeller reveals the exact edges of the graph used during the inference time. Therefore, GCN and similar GNN architectures leak edges to LinkTeller since the influence scores are propagated through edges of the graph in such architectures. We also consider a more general attack, LPA, that is not tuned to any specific architecture given by He et al. [22]. LPA has many attack scenarios based on the partial knowledge available with the attacker. Here, we choose the one that is most applicable in our setting, i.e., the attacker has access to only the inference node features $\mathbf{F}^I$. Specifically, LPA uses the features and the obtained embeddings $\mathbf{L}^I$ to predict the likelihood of having an edge. Nodes with more similar embeddings are ranked higher for the likelihood of having an edge. We reproduce both of these attacks successfully by following their protocols exactly.

Attack procedure and metrics. We follow the same attack procedures set by the previous works [22, 59]. For the transductive setting, we sample a set of 500 edges and 500 non-edges to measure how well the attacks can separate the edges from non-edges. For the inductive setting, we sample a set of 500 inference nodes $\mathcal{V}^I$ to create a subgraph that has not been trained on and then we evaluate the attacks on all the edges and non-edges in the subgraph. Further, LinkTeller measures the performance on high and low-degree nodes separately to understand how their attack performs on nodes with different degrees. They find out that the attack performs better on high-degree nodes. We also evaluate the attack on these additional scenarios on the datasets they evaluate and report the values in the Appendix C.2. We use the Area Under Receiver Operating Curve (AUC) metric to measure the attack performance as done in the LPA and LinkTeller works. This metric has also been extensively used for measuring inference attack performance on other kinds of private data in prior work [9, 22, 43, 59]. For graph data, the current methodology to measure AUC asks how well an attack can identify a randomly sampled existing edge from a non-edge. Higher AUC is typically interpreted as the attack being more successful in identifying the edges used by the model, i.e., the model is less resilient to the attack. We report the average performance of each attack across 5 seeds.

Table 2: Node classification results for the transductive setting using micro-average F1 scores when the models are not trained with DP. LPGNet significantly outperforms MLP, in all datasets, which confirms the generality of our key ideas.

Dataset	MLP	LPGNet-1	LPGNet-2	GCN
Facebook	0.74 ± 0.01	0.83 ± 0.01	0.87 ± 0.01	0.93 ± 0.01
Citeseer	0.60 ± 0.01	0.65 ± 0.01	0.67 ± 0.01	0.72 ± 0.0
Cora	0.60 ± 0.01	0.69 ± 0.02	0.73 ± 0.01	0.81 ± 0.0
PubMed	0.73 ± 0.01	0.75 ± 0.0	0.75 ± 0.0	0.79 ± 0.0

5.2 Results: Transductive Settings

We will denote LPGNet with *nl* additional MLP as LPGNet-*nl*.

Utility in the non-DP setting. We compare the utility offered by LPGNet to that of MLP and GCN in the non-DP setup to gauge how well our key observations in Section 3 generalize to all datasets. In Table 2 we present the F1 scores of the best parameter-tuned models. First, observe that the GCN outperforms the MLP by 6-19% across all datasets. Second, LPGNet-1 outperforms the MLP as well by 2-9% with just one additional stack layer. With every additional stack layer, LPGNet's performance improves across all datasets with LPGNet-3 (not shown here) performing 4-15% better than MLP and only 4-6% worse than the GCN, respectively. This shows that our idea of iteratively improving the homophily signal in the clusters captured by MLP by using cluster degree vectors works well on all datasets in the transductive setting. Note that LPGNet achieves this performance using the coarse-grained cluster degree vectors instead of using fine-grained edge information as in GCN.

Utility in the DP setting. When we add noise, we observe that LPGNet's utility is not affected as much as that of DpGCN models. Consequently, on all datasets in the transductive setting, LPGNet performs better than MLP and the DpGCN at a majority of privacy budgets $\epsilon \in [1, 10]$ we evaluate. We compare DpGCN with only LPGNet-1 and LPGNet-2 as the privacy budgets increase with every additional stack layer used in LPGNet. Figure 5 shows the comparison between LPGNet, MLP, and DpGCN. LPGNet models perform better than the MLP on all datasets for $\epsilon \geq 2$. In contrast, DpGCN performs up to 2.6× worse than LPGNet at $\epsilon \leq 2$. LPGNet provides better utility than DpGCN for $2 < \epsilon < 7$ on all datasets as

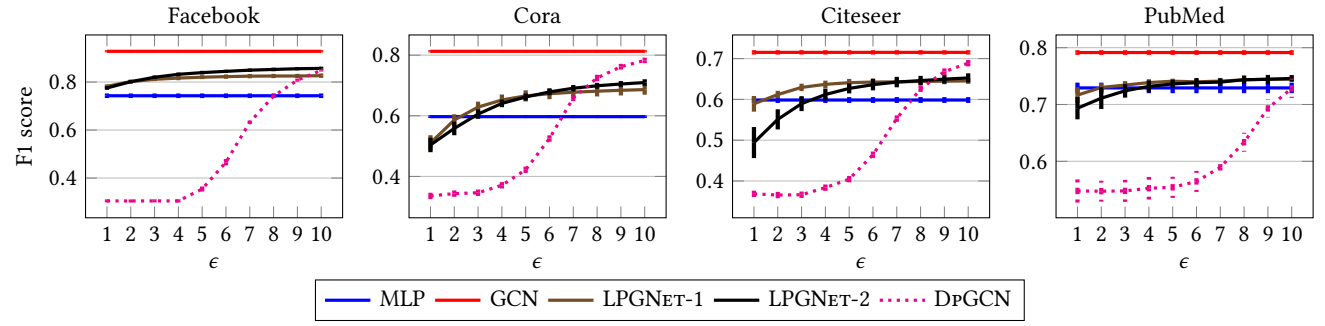

Figure 5: Utility of DP models in the transductive setting for various ϵ.

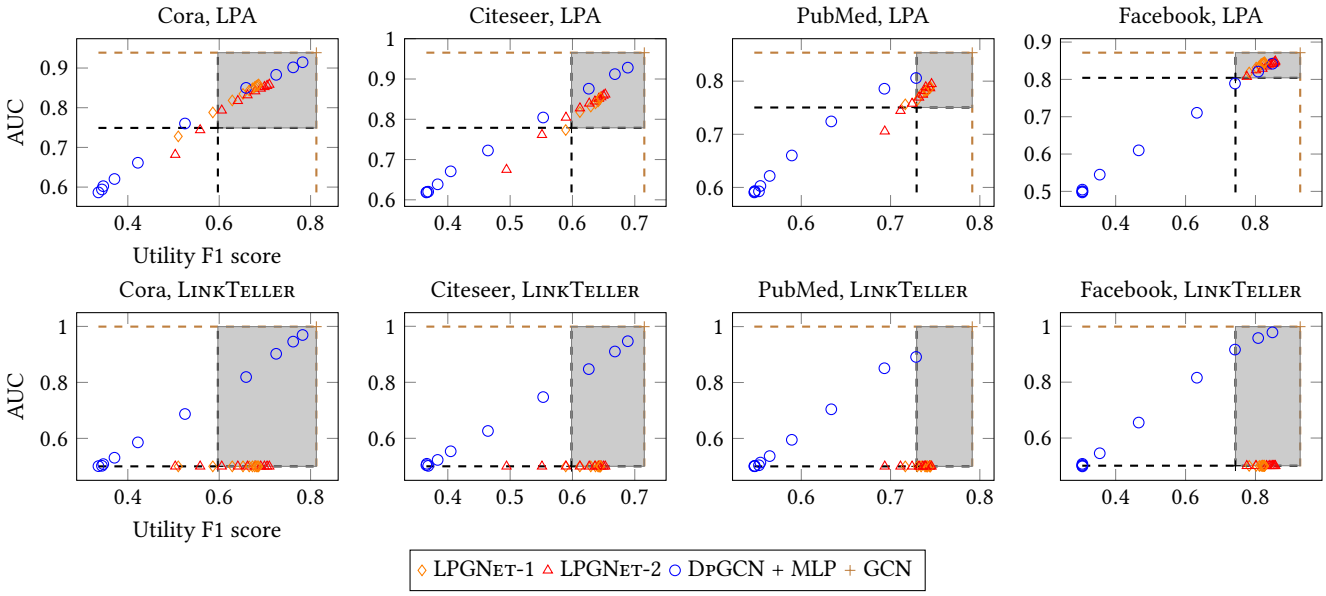

Figure 6: Attack AUC vs Utility F1-score in the transductive setting for two state-of-the-art attacks. If we fix the utility level, LPGNet models always have better resilience to attacks than the corresponding DpGCN model with that utility.

well. With $\epsilon \geq 7$, LPGNet performs better than DpGCN on 2 out of the 4 evaluated datasets.

Result 1: LPGNet outperforms MLP for all $\epsilon \geq 2$ and up to 2.6× better than DpGCN for relatively low $\epsilon \in [1, 7]$ on all datasets.

Privacy resilience measured via attacks. The privacy budget ϵ is a theoretical upper bound on the amount of privacy leaked by a DP-trained model. Therefore, it may not be meaningful to say that two models offer same level of actual privacy if trained with same ϵ. To estimate the true privacy resilience, we evaluate how well state-of-the-art link stealing attacks, LinkTeller and LPA, perform on LPGNet and DpGCN. Attack-based evaluations are common for other DP systems as well [24, 39].

Figure 6 shows the tradeoffs with utility (classification accuracy) plotted on the x-axis and attack accuracy on y-axis. Each point on the plot is for a distinct choice of model and ϵ. The performance of non-DP models, namely vanilla MLP and GCN, are baselines to compare DP solutions with and shown with dashed lines. We expect the attack resilience to be the best (lowest attack accuracy) for MLP since it does not use edges at all. We also expect (vanilla) GCN to offer the best utility since it uses the full raw edge structure with no noise. We color in grey the sweet spot region on the plot, where the utility is better than MLP but worse than that of GCN, and the attack resilience better than GCN but worst than MLP.

Figure 6 shows that, compared to DpGCN, LPGNet's models are tightly concentrated in the sweet spot region and more towards the south-east side of that region. This implies that its utility is closer to GCN and attack AUC is closer to MLP.

We zoom in to the only 10 configurations where DpGCN has better utility than MLP and compare its attack AUC with LPGNet. We provide the results in Appendix C.1, Table 6. LPGNet outperforms DpGCN in both utility and attack resilience in Facebook and PubMed datasets. For Cora and Citeseer, the attack resilience of LPGNet is always better than DpGCN. The best attack has an AUC

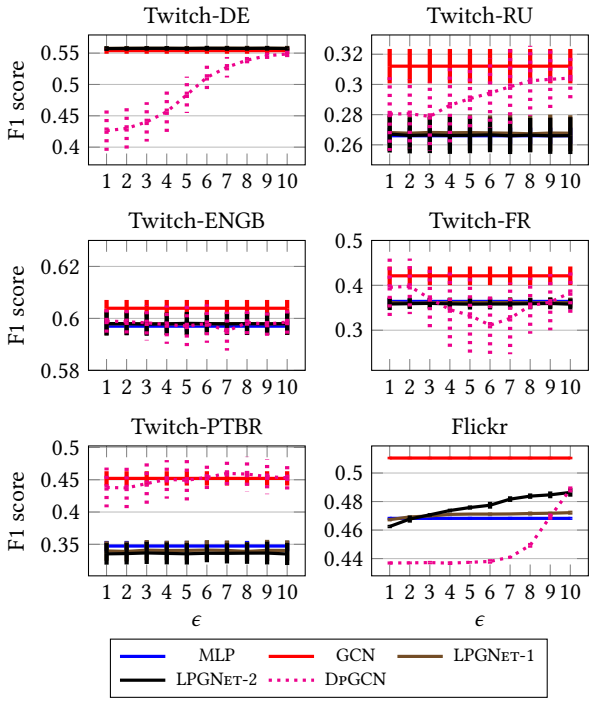

Figure 7: Utility of DP models in the inductive settings.

of 0.85-0.98 on DPGCN which is about 10-22% higher than the AUC on MLP in absolute difference. In contrast, the best attack has an AUC of 0.79-0.86 on LPGNET which is about 4-11% higher than the AUC on MLP. Thus, LPGNET offers better privacy-utility tradeoffs compared to DPGCN *consistently* across all datasets.

The AUCs already show that LPGNET leaks less edge information than DPGCN. However, the AUC of 0.86 for LPGNET is still high. So, does that mean LPGNET is leaking a lot of edge information? In short, the answer is *No* and we discuss this in Section 5.4.

Result 2: LPGNET often hits the sweet spots and offers significantly better attack resilience than DPGCN at similar utility.

Due to space constraints, the full utility and attack resilience data for different ϵ values is left to the Appendix C.1.

5.3 Results: Inductive Setting

The inductive setting is challenging for many state-of-the-art GNN architectures as they have to transfer their knowledge to a different graph [20, 28]. Here, we measure how well LPGNET generalizes to unseen graphs. Recall that we have a different evaluation procedure for both utility and attack performance in this setting (see Section 5).

Utility in the non-DP setting. We observe that all the models perform significantly worse than in the transductive setting and their performances are very close to each other as well. The vanilla GCN performs only 4-11% better than the MLP. In the Twitch datasets, LPGNET is either on-par or improves by 1% over MLP in all of its configurations (5 datasets ×2 LPGNET architectures). For Flickr dataset, LPGNET improves over MLP by 1-3% when GCN itself is 4% better than LPGNET. Therefore, even in the inductive setting, LPGNET is a better choice than MLP utility-wise as it is on-par or bridges the gap between MLP and GCN across all datasets.

Utility in the DP setting. For all $\epsilon \in [1, 10]$, we observe that LPGNET-2 has better utility than MLP in Flickr and is on-par with MLP in Twitch. Figure 7 shows utility of LPGNET and DPGCN models. LPGNET-1's performance changes at most by 1% at all levels of ϵ and LPGNET-2 performs better than LPGNET-1 by up to 2% across all datasets. In 4 out of 6 datasets, LPGNET-2 performs on-par or better than DPGCN at all ϵs, therefore, achieving better utility at the same theoretical privacy guarantee. LPGNET models have worse utility than DPGCN models for Twitch-PTBR and Twitch-RU.

On Twitch datasets, LPGNET's lack of improvement over MLP can be explained due to less useful information obtained from the cluster degree vectors. In Twitch, every node has similar number of neighbors from both clusters (on average) making it difficult to distinguish between the degree vectors of nodes from both clusters. For instance, nodes from cluster-1 in Twitch-RU have 75% of their neighbors from cluster-1 and 25% neighbors from cluster-2 as expected where as nodes from cluster-2 also have 70% of their neighbors from cluster-1 and only 30% from cluster-2. We observe a similar pattern for all Twitch datasets. This lack of homophily affects all architectures and LPGNET is the most affected since it depends on cluster degree vectors rather than the exact edge information. Nevertheless, even with DP noise, LPGNET reliably performs at least as good as MLP unlike DPGCN.

Result 3: LPGNET-2 reliably performs on-par or better than MLP for both Flickr and Twitch datasets.

Privacy resilience measured via attacks. We first observe that LPGNET has good resilience against both attacks, i.e., attack AUC is lesser than 0.65 across all evaluated configurations. DPGCN, in contrast, has significantly worse attack resilience than LPGNET for Twitch and Flickr datasets with best attack AUC often reaching up to 0.98 even at the same utility level as LPGNET. In 2 out of 5 Twitch datasets where DPGCN has better utility than LPGNET the attack AUC is above 0.9 for most ϵs whereas the attack AUC on LPGNET is almost 0.5. For Flickr, we zoom in on cases where *both* LPGNET and DPGCN have utility better than MLP (see Table 11 in Appendix C.2). LPGNET offers *significantly* better resilience against attacks than the corresponding DPGCN with the same level of utility. The best attack on LPGNET has an AUC of 0.56, at $\epsilon = 10$, but 0.90 on DPGCN (34% worse than LPGNET). Therefore, our Result 2 from Section 5.2 is true for the inductive setting as well.

We provide the detailed results including utility and attack AUCs of all configurations for the inductive setting in the Appendix C.2.

5.4 Discussion: Interpreting the Results

We explain some implications of our findings which may not be immediately obvious from knowledge of prior attacks [22, 59].

5.4.1 LPA has relatively high AUC against LPGNET. So, is LPGNET leaking the edges against LPA?. The answer depends on the nature of the attack and its evaluation strategy. In prior work, the attack AUC is evaluated on the ability to distinguish a randomly sampled edge from a non-edge. The AUC can be high even when no individual edges are used (memorized) by the trained model. For example,

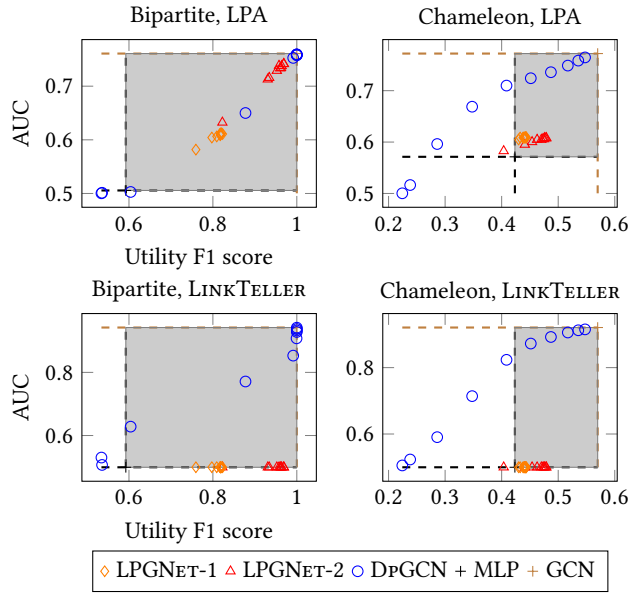

Figure 8: Attack AUC vs Utility in the transductive setting for Bipartite **and** Chameleon **datasets, with low homophily.**

LPA's high AUC is an artifact of the distribution of the graphs that are being learned by the GNNs. To understand this, consider homophilous graphs which have 80% of their edges connecting nodes from the same cluster and only 20% of the time there is no edge connecting nodes within the same cluster. If the clusters learned by a GNN are revealed, then it is easy to guess which nodes are more likely to have an edge between them. A simple "attack" would predict that an edge exists between two nodes within the same cluster, otherwise not, and have high AUC. This case would arise even if an MLP, that uses just node features, discovers these clusters. LPGNet (or any edge-DP mechanism) does not prevent such attacks that predict edges because they belong to a distribution that is being learned by the ML model. To confirm that LPA is our simple attack, we take the top-$|\mathcal{E}|$ edges as predicted by LPA for GCN trained on the Cora dataset which has the aforementioned distribution of edges and non-edges. Almost 99% of its edge predictions were intra-cluster. Unlike LPA, LinkTeller's high AUC is an artifact of its ability to tell whether a specific edge is in the training graph irrespective of the distribution of node features and edges. Therefore, a high AUC for LinkTeller suggests that the model is responsible for leaking the edges used for training or inference. It is thus expected that LinkTeller's AUC on MLP should be 0.5. LPGNet has the edge-DP guarantee, so it protects against such attacks when the model uses individual edges for training and inference.

In summary, it is misleading to conclude that GNNs surely leak individual edges based on just the attack's AUC; instead, we must take into account the attack details (e.g., LPA vs. LinkTeller).

5.4.2 Why is MLP a good baseline? When we compute the attack AUC, as we explained, it is possible to get a high score whether or not the edges have been used by the GNN. [6] If we want to marginally compute the effect of using the edges in training then we would have to *causally* remove the edges from training and measure the attack performance. MLP offers that baseline—it uses no edges. Therefore, the increase in attack AUC on a GNN vs MLP indicates how much a GNN leaks about the edges which the attacker already does not know from its background knowledge about node features and the distribution of graphs to which the training graph belongs. For LPGNet, the increase in attack AUC over MLP is smaller than in other architectures (see Figure 6).

5.4.3 Can LPA have low AUC on MLP and LPGNet? Indeed, this is possible. Consider graphs where the node features do not correlate well with the clusters. In such graphs, the AUC of LPA on MLP can be close to 0.5 but the classification accuracy would be low as well. However, LPGNet can have a high classification accuracy while having a low LPA attack AUC. LPGNet, in this case, will almost entirely rely on the edge structure to improve the classification. We show this phenomena using two datasets. We first construct a synthetic bipartite graph, inspired by dating networks, so that the node features are not very correlated with the graph structure. We then take a real-world citation network (based on Wikipedia) that is popularly used as a challenge dataset for evaluating GNNs for its lack of homophily [42, 47, 66]. LPGNet has low LPA AUC but higher classification accuracy than MLP in these examples. We provide more details about the datasets in Appendix C.3.

In Figure 8, we provide the attack AUC vs utility plots for these two datasets. Observe that in both datasets, the AUC of best attack on MLP is close to 0.5. LPGNet has better utility than MLP in almost all evaluated configurations. In the Bipartite graph, LPGNet-1 itself is better than MLP by $16-22\%$. In the Chameleon graph, LPGNet offers up to 6% better utility than MLP. LPGNet performs well on these datasets because two nodes within the same cluster are more likely to have similar distribution of labels in their neighborhood, hence similar cluster degree vectors. For instance, in the Bipartite graph, nodes of same cluster always have neighbors from the opposite cluster even though, by definition, it is not homophilous. LPGNet design can work for such non-homophilous graphs.

The best attack on LPGNet-1 has at most 0.61 AUC on both datasets. The best attack on LPGNet-2 has an AUC of 0.74 in the Bipartite graph and 0.61 in the Chameleon graph. The attacks perform better on DpGCN than LPGNet and the best attack has close to 0.9 AUC on DpGCN even at the same utility level as LPGNet.

5.4.4 How to prevent edge leakage from node attributes? It is natural to be concerned about how the node attributes themselves can leak edge information. However, the definitions of edge-DP or node-DP are not designed to address such privacy concerns. In our problem setup, the edge leakage from attributes corresponds to a different notion of privacy which asks for bounding the public information known to an attacker from knowing the attributes beforehand. One way to address this kind of leakage is to change the application setup itself. For instance, if an application allows users to store both of the node attributes and edges locally, then better privacy can be expected by using local DP for attributes and edges. Such setups are important future work and there is no existing work that designs DP algorithms for them, to the best of our knowledge.

[6] The precision and recall will be similar to AUC for this evaluation methodology.

6 RELATED WORK

In this paper, we provide a differentially private learning technique for node classification on private graph-structured data.

Semi-supervised learning from graph data. Semi-supervised learning techniques on graphs combine the knowledge obtained from the graph structure and the node features for various applications such as node classification. Graph neural networks have become the defacto standard to perform semi-supervised learning on graphs. Initial GNN architectures, inspired from recurrent neural networks [3], apply the same parameterized function over node features recursively. The parameterized function performs a weighted average over the features of a node's neighbors to update its features [19, 50]. Current GNN architectures are inspired from convolutional neural networks [32] where the input features are passed through *different* parameterized functions (convolutions) to compute the outputs. They mainly fall into two categories called spatial and spectral, depending on the nature of parameterized function used. The function could perform a weighted average over the features of neighboring nodes for each node (spatial) [7, 18, 20, 53, 62] or select the important features using the spectral decomposition of the graph Laplacian (spectral) [13, 23, 28]. Refer to this extensive survey for existing GNN architectures [60]. In all these architectures, the adjacency matrix of the graph, i.e., the knowledge of exact neighbors is essential. Therefore, LinkTeller is expected to perform well on them as demonstrated on two architectures, GCNs and graph attention networks [59]. Further, in Sections 2 and 5.2 we show that noising the adjacency matrix using DpGCN will adversely affect their utility due to severely perturbed graph structure.

Our idea is to move away from the traditional GNN architectures and not use the adjacency matrix for propagation in intermediate layers. Rather, we propose to use only MLPs for node classification and use the graph structure in the form of cluster degree vectors as their input features to iteratively improve their classification performance. Our idea is in the same spirit of recent works that design purely MLP-based architectures to achieve comparable performance to state-of-the-art nets in vision [36, 52].

Attacks on graph neural networks. Attacks on graph neural networks to steal (infer) the edges is a recent phenomena. Duddu et al. [15] showed several attacks to predict the membership of both nodes and links with blackbox and whitebox [7] access to train the GCNs. The attack for identifying links uses another graph from the same distribution and node embeddings obtained from the victim GCN. He et al. [22] showed link stealing attacks with blackbox access to a trained GCN (LPA) and more recently, Wu et al. [59] showed, LinkTeller, a stronger attack with blackbox access to the learned GCN and without needing access to features of nodes used in training. We have discussed these attacks in Section 5. Other attacks on GNNs have focused on poisoning the training dataset to affect the final classification results [5, 10].

Existing defenses against the attacks. DpGCN is the only defense which provides an edge-DP guarantee against the aforementioned attacks for any graph. Zhou et al. [65] proposes training a GCN in the federated setup but only adds noise after aggregating the features from the neighbors which is not sufficient for differential privacy. Wu et al. [58] proposes using private training of GNN for recommendations, but they model only bipartite graphs of user-item edges and their techniques do not extend to general graphs. A concurrent work proposes to train a GNN with node-DP guarantee which may require high ϵs for our setup as shown by the high privacy budgets used in that work ($\epsilon > 10$) to get a utility better than an MLP [11]. Further, they provide their privacy analysis only for single layer GCNs and do not evaluate the state-of-the-art attacks. Therefore, it is not clear if their models provide any privacy resilience against the attacks.

Differentially private queries on graphs. For the edge-DP setup, many works design differentially private algorithms for estimating degree distributions [4, 26, 41], subgraph counts [41], synthetic graphs [33, 37, 48, 55, 56, 61] and communities [25, 38, 40]. Similarly, for the node-DP setup, there are algorithms for computing degree distributions [12, 45] and subgraph counts [8, 14]. All of the aforementioned algorithms are in the centralized DP setup where a central server is trusted. In the LDP setup where there is no trusted centralized server, there are edge-LDP algorithms for estimating aggregate statistics on graphs [57], community detection [44], and hierarchical clustering [30]. Our proposed technique can be implemented in both the centralized and local setups since computing cluster degree vectors is an operation that requires only the knowledge of one's neighbors and their cluster labels.

7 CONCLUSION

We have presented LPGNet, a new stacked neural network architecture for learning graphs with privacy-sensitive edges. LPGNet is carefully designed to strengthen the property of homophily, when present in the graph. LPGNet provides differential privacy for edges. It exhibits meaningful utility-privacy tradeoffs compared to other existing architectures that either use edge-DP or are trivially edge-private in most evaluated datasets and configurations.

ACKNOWLEDGMENTS

We thank Aneet Kumar Dutta and Bo Wang for helping us with parsing the results and generating tables for an earlier version of this paper. We are grateful for the constructive feedback from the anonymous reviewers and for conducting the revision procedure. We thank Kunwar Preet Singh for his help while revising the paper. We thank Crystal Center and its sponsors, Singapore Ministry of Education (MoE), National University of Singapore (NUS), and National Research Foundation Singapore (NRF) for their generous grants. Aashish Kolluri is supported by Crystal Centre under the Grant No.: E-251-00-0105-01 and MoE under the Grant No.: A-0008530-00-00. Teodora Baluta is supported by NRF under its NRF Fellowship Programme NRF-MRFFAI1-2019-0004 and MoE under the Grant No.: T2EP20121-0011. Teodora Baluta is also supported by the Google PhD Fellowship. Bryan Hooi is supported in part by NUS ODPRT grant under the Grant No.: R252-000-A81-133.

REFERENCES

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

[7] Only the node embeddings at intermediate layers are visible

[2] Jasmijn Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Sima'an. 2017. Graph Convolutional Encoders for Syntax-aware Neural Machine Translation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*.

[3] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks* (1994).

[4] Jeremiah Blocki, Avrim Blum, Anupam Datta, and Or Sheffet. 2013. Differentially private data analysis of social networks via restricted sensitivity. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*.

[5] Aleksandar Bojchevski and Stephan Günnemann. 2019. Adversarial attacks on node embeddings via graph poisoning. In *International Conference on Machine Learning*.

[6] Kamalika Chaudhuri, Claire Monteleoni, and Anand D Sarwate. 2011. Differentially private empirical risk minimization. *Journal of Machine Learning Research* (2011).

[7] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247* (2018).

[8] Shixi Chen and Shuigeng Zhou. 2013. Recursive mechanism: towards node differential privacy and unrestricted joins. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*.

[9] Christopher A Choquette-Choo, Florian Tramer, Nicholas Carlini, and Nicolas Papernot. 2021. Label-only membership inference attacks. In *International Conference on Machine Learning*.

[10] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. 2018. Adversarial attack on graph structured data. In *International Conference on machine learning*.

[11] Ameya Daigavane, Gagan Madan, Aditya Sinha, Abhradeep Guha Thakurta, Gaurav Aggarwal, and Prateek Jain. 2021. Node-Level Differentially Private Graph Neural Networks. *arXiv preprint arXiv:2111.15521* (2021).

[12] Wei-Yen Day, Ninghui Li, and Min Lyu. 2016. Publishing graph degree distribution with node differential privacy. In *Proceedings of the 2016 International Conference on Management of Data*.

[13] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems* (2016).

[14] Xiaofeng Ding, Xiaodong Zhang, Zhifeng Bao, and Hai Jin. 2018. Privacy-preserving triangle counting in large graphs. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*.

[15] Vasisht Duddu, Antoine Boutet, and Virat Shejwalkar. 2020. Quantifying Privacy Leakage in Graph Embedding. *arXiv preprint arXiv:2010.00906* (2020).

[16] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science* (2014).

[17] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph neural networks for social recommendation. In *The World Wide Web Conference*. 417–426.

[18] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. 2018. Large-scale learnable graph convolutional networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

[19] Marco Gori, Gabriele Monfardini, and Franco Scarselli. 2005. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*

[20] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*.

[21] Michael Hay, Chao Li, Gerome Miklau, and David Jensen. 2009. Accurate estimation of the degree distribution of private networks. In *2009 Ninth IEEE International Conference on Data Mining*.

[22] Xinlei He, Jinyuan Jia, Michael Backes, Neil Zhenqiang Gong, and Yang Zhang. 2021. Stealing links from graph neural networks. In *USENIX Security Symposium*.

[23] Mikael Henaff, Joan Bruna, and Yann LeCun. 2015. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163* (2015).

[24] Bargav Jayaraman and David Evans. 2019. Evaluating differentially private machine learning in practice. In *USENIX Security Symposium*. 1895–1912.

[25] Tianxi Ji, Changqing Luo, Yifan Guo, Jinlong Ji, Weixian Liao, and Pan Li. 2019. Differentially private community detection in attributed social networks. In *Asian Conference on Machine Learning*.

[26] Vishesh Karwa and Aleksandra B Slavković. 2012. Differentially private graphical degree sequences and synthetic graphs. In *International Conference on Privacy in Statistical Databases*.

[27] Shiva Prasad Kasiviswanathan, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. 2013. Analyzing graphs with node differential privacy. In *Theory of Cryptography Conference*.

[28] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[29] Aashish Kolluri, Teodora Baluta, Bryan Hooi, and Prateek Saxena. 2022. LPGNet: Link Private Graph Networks for Node Classification. *arXiv preprint arXiv:2205.03105* (2022).

[30] Aashish Kolluri, Teodora Baluta, and Prateek Saxena. 2021. Private Hierarchical Clustering in Federated Networks. *arXiv preprint arXiv:2105.09057* (2021).

[31] Oliver Lange and Luis Perez. 2020. Traffic prediction with advanced Graph Neural Networks. https://deepmind.com/blog/article/traffic-prediction-with-advanced-graph-neural-networks

[32] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* (1998).

[33] Wentian Lu and Gerome Miklau. 2014. Exponential random graph estimation under differential privacy. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*.

[34] Yao Ma, Xiaorui Liu, Tong Zhao, Yozen Liu, Jiliang Tang, and Neil Shah. 2021. *A Unified View on Graph Neural Networks as Graph Signal Denoising*.

[35] Miller McPherson, Lynn Smith-Lovin, and James M Cook. 2001. Birds of a feather: Homophily in social networks. *Annual review of sociology* (2001).

[36] Luke Melas-Kyriazi. 2021. Do you even need attention? a stack of feed-forward layers does surprisingly well on imagenet. *arXiv preprint arXiv:2105.02723* (2021).

[37] Darakhshan Mir and Rebecca N Wright. 2012. A differentially private estimator for the stochastic kronecker graph model. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops*.

[38] Yvonne Mülle, Chris Clifton, and Klemens Böhm. 2015. Privacy-Integrated Graph Clustering Through Differential Privacy.. In *EDBT/ICDT Workshops*.

[39] Milad Nasr, Shuang Song, Abhradeep Thakurta, Nicolas Papernot, and Nicholas Carlini. 2021. Adversary instantiation: Lower bounds for differentially private machine learning. *arXiv preprint arXiv:2101.04535* (2021).

[40] Hiep H Nguyen, Abdessamad Imine, and Michaël Rusinowitch. 2016. Detecting communities under differential privacy. In *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society*.

[41] Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. 2007. Smooth sensitivity and sampling in private data analysis. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*.

[42] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. 2020. Geom-gcn: Geometric graph convolutional networks. *arXiv preprint arXiv:2002.05287* (2020).

[43] Apostolos Pyrgelis, Carmela Troncoso, and Emiliano De Cristofaro. 2018. Knock Knock, Who's There? Membership Inference on Aggregate Location Data. *ArXiv* abs/1708.06145 (2018).

[44] Zhan Qin, Ting Yu, Yin Yang, Issa Khalil, Xiaokui Xiao, and Kui Ren. 2017. Generating synthetic decentralized social graphs with local differential privacy. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*.

[45] Sofya Raskhodnikova and Adam Smith. 2015. Efficient lipschitz extensions for high-dimensional graph statistics and node private degree distributions. *arXiv preprint arXiv:1504.07912* (2015).

[46] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. 2021. Multi-Scale Attributed Node Embedding. *Journal of Complex Networks* (2021).

[47] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. 2021. Multi-scale attributed node embedding. *Journal of Complex Networks* (2021).

[48] Alessandra Sala, Xiaohan Zhao, Christo Wilson, Haitao Zheng, and Ben Y Zhao. 2011. Sharing graphs using differentially private graph models. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*.

[49] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. 2020. Superglue: Learning feature matching with graph neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*.

[50] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE transactions on neural networks* (2008).

[51] Yantao Shen, Hongsheng Li, Shuai Yi, Dapeng Chen, and Xiaogang Wang. 2018. Person re-identification with deep similarity-guided graph neural network. In *Proceedings of the European conference on computer vision (ECCV)*.

[52] Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, et al. 2021. Mlp-mixer: An all-mlp architecture for vision. *arXiv preprint arXiv:2105.01601* (2021).

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

[54] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*.

[55] Yue Wang and Xintao Wu. 2013. Preserving differential privacy in degree-correlation based graph generation. *Transactions on data privacy* (2013).

[56] Yue Wang, Xintao Wu, and Leting Wu. 2013. Differential privacy preserving spectral graph analysis. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*.

[57] Chengkun Wei, Shouling Ji, Changchang Liu, Wenzhi Chen, and Ting Wang. 2020. Asgldp: Collecting and generating decentralized attributed graphs with

local differential privacy. *IEEE Transactions on Information Forensics and Security* (2020).

[58] Chuhan Wu, Fangzhao Wu, Yang Cao, Yongfeng Huang, and Xing Xie. 2021. Fedgnn: Federated graph neural network for privacy-preserving recommendation. *arXiv preprint arXiv:2102.04925* (2021).

[59] Fan Wu, Yunhui Long, Ce Zhang, and Bo Li. 2021. LinkTeller: Recovering Private Edges from Graph Neural Networks via Influence Analysis. *arXiv preprint arXiv:2108.06504* (2021).

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

[61] Qian Xiao, Rui Chen, and Kian-Lee Tan. 2014. Differentially private network data release via structural inference. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*.

[62] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How powerful are graph neural networks? *ICLR* (2019).

[63] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

[64] Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. 2019. T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE Transactions on Intelligent Transportation Systems* (2019).

[65] Jun Zhou, Chaochao Chen, Longfei Zheng, Xiaolin Zheng, Bingzhe Wu, Ziqi Liu, and Li Wang. 2020. Privacy-preserving graph neural network for node classification. *arXiv e-prints* (2020), arXiv–2005.

[66] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. 2020. Beyond homophily in graph neural networks: Current limitations and effective designs. *NeurIPS* (2020).