LINKTELLER: Recovering Private Edges from Graph Neural Networks via Influence Analysis

Fan Wu[1] Yunhui Long[1] Ce Zhang[2] Bo Li[1]
[1]University of Illinois at Urbana-Champaign [2]ETH Zürich
{fanw6,ylong4,lbo}@illinois.edu ce.zhang@inf.ethz.ch

***Abstract*—Graph structured data have enabled several successful applications such as recommendation systems and traffic prediction, given the rich node features and edges information. However, these high-dimensional features and high-order adjacency information are usually heterogeneous and held by different data holders in practice. Given such vertical data partition (*e.g.*, one data holder will only own either the node features or edge information), different data holders have to develop efficient joint training protocols rather than directly transferring data to each other due to privacy concerns. In this paper, we focus on the *edge privacy*, and consider a training scenario where the data holder Bob with node features will first send training node features to Alice who owns the adjacency information. Alice will then train a graph neural network (GNN) with the joint information and provide an inference API to Bob. During inference time, Bob is able to provide test node features and query the API to obtain the predictions for test nodes. Under this setting, we first propose a privacy attack LINKTELLER via influence analysis to infer the private edge information held by Alice via designing adversarial queries for Bob. We then empirically show that LINKTELLER is able to recover a significant amount of private edges in different settings, both including inductive (8 datasets) and transductive (3 datasets), under different graph densities, significantly outperforming existing baselines. To further evaluate the privacy leakage for edges, we adapt an existing algorithm for differentially private graph convolutional network (DP GCN) training as well as propose a new DP GCN mechanism LAPGRAPH based on Laplacian mechanism to evaluate LINKTELLER. We show that these DP GCN mechanisms are not always resilient against LINKTELLER empirically under mild privacy guarantees ($\varepsilon > 5$). Our studies will shed light on future research towards designing more resilient privacy-preserving GCN models; in the meantime, provide an in-depth understanding about the tradeoff between GCN model utility and robustness against potential privacy attacks.**

***Index Terms*—Graph Neural Networks, Edge Privacy Attack**

I. INTRODUCTION

Graph neural networks (GNNs) have been widely applied to different domains owing to their ability of modeling the high-dimensional feature and high-order adjacency information on both homogeneous and heterogeneous graph structured data [1]–[3]. The high-quality graph structured data have enabled a range of successful applications, including traffic prediction [4], recommendation systems [5], and abnormal access detection [6]. As these applications are becoming more and more prevalent, privacy concerns in these applications are non-negligible given the sensitive information in the graph data. Thus, undesirable outcomes may arise due to lack of understanding of the models and application scenarios.

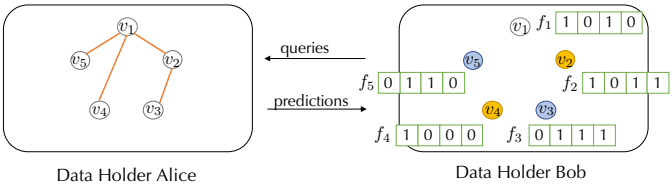


Fig. 1: Vertically partitioned graph data for different data holders.

In this paper, we aim at understanding the *edge privacy* in applications of GNN models. We focus on one specific scenario as *training/serving GNN models in a vertically partitioned graph setting*. As illustrated in Figure 1, in this setting, node features and adjacency information (edges) are isolated or hosted by different data holders. Our interest in this setting is inspired not only by recent academic research (*e.g.*, Zhou *et al.* [3] proposed a privacy-preserving GNN (PPGNN) mechanism via homomorphic encryption under this setting) but also a real-world industrial example we see and the potential privacy risks it incurs. In such an example, an international Internet company hopes to train a single GNN model jointly using data collected by two of its subdivisions (namely, data holder Alice and Bob). Because these subdivisions focus on different products, their data are heterogeneous in nature. Specifically, in this example, Alice collects user interaction (social network) data (*i.e.*, adjacency information), and Bob collects user behavior data (*i.e.*, node features). Noticing the potential benefit of integrating user interaction data into its predictive model, Bob hopes to enrich the model using data collected by Alice. Although they belong to the same company, directly copying the user interaction data from Alice to Bob is not allowed due to privacy concerns. Thus, Bob will first send training data containing node features and labels to Alice, and Alice will train a GNN model jointly with her edge information. Then Alice will release an inference API to Bob. During inference, Bob would send a new set of test nodes with their features to query the API and obtain corresponding predictions. Different users can query the API to enjoy the service from Alice. For instance, in practice there are several ML/AI service platforms that provide similar interactions—taking the training data from users to train an ML model and providing inference APIs for users to make queries with their test data—such as Vertex AI [7] from Google Cloud, ParlAI platform [8] from Facebook, and InfoSphere Virtual Data Pipeline [9] from IBM.

During this type of interaction, *the fundamental question is to understand the risks of edge privacy (mainly for data*

holder Alice) for training and releasing a GNN inference API on graph data, as well as possible ways to amortise such risks.

Challenges and Problem Formulation. The main motivation and challenge of the problem attributes to the heterogeneity of data—one data holder owns the features of users (*i.e.*, node features), while the other holds the "connections" or "interactions" among users (*i.e.*, adjacency information) as shown in Figure 1. Inspired by this real-world example, we abstract it into the following technical problem. Let there be n users and $A \in \{0,1\}^{n\times n}$ be the adjacency information. Data holder Alice has full access to adjacency information A while it is kept secret from the data holder Bob. Bob interacts with Alice during both *training* and *inference* stages.

1) Training: During training, Bob (or some other users) collects (training) node features and labels for a subset of users, forming a feature matrix X with label vector y, and sends them to Alice. Alice then trains a GNN model using all the node features from Bob and her collected adjacency information A, and releases an inference API to Bob.
2) Inference: During inference, Bob collects features for another (test) subset of users X', and sends them to Alice via the inference API, who will run inference using the trained GNN model, and return corresponding predictions.

Given this interaction model, we aim to ask: Whether the inference API will leak private information of the adjacency information to a potentially malicious user Bob indirectly? How can we better protect the adjacency information from privacy leakage while preserving high model utility?

Apart from this specific case, there have been similar concerns from different real-world cases. For instance, the advertisement department of Facebook would usually hold certain public features of individuals (*i.e.*, node features), and needs to query the predictions from another department that holds the social network connection information which is private. Thus, how to protect the edge privacy in this setting is critical. However, directly conducting such privacy attacks is challenging. For instance, given a large graph, naively comparing the similarities between nodes to infer their connections is clearly not enough. On the other hand, it is known that the trained GNN is based on the node influence propagation [10]: If two nodes are connected, there is a high chance that changing the features of one would affect the prediction of the other. Thus, we hope to address the research question: *Is it possible to design an effective edge re-identification attack against GNNs based on the node influence information?*

Different from existing work [11] which collects node pairs with and without connections to train a model to infer the existence of an edge, in this paper, we aim to analyze and leverage the *node influence* to predict potential edge connections. In particular, we first propose an attack strategy LINKTELLER under such a vertically data partitioned setting based on the node influence analysis, and explore how much the private adjacency information could be revealed from Alice via answering queries from Bob. Then we will evaluate the proposed LINKTELLER attack against both an existing and a proposed differentially private graph convolutional network (DP GCN) mechanisms to analyze whether the LINKTELLER could further attack the privacy preserving GCN models. In addition, we explore what is the *safe* privacy budget to choose in order to protect the trained GCN models from being attacked by privacy attacks such as LINKTELLER on different datasets via extensive empirical evaluation.

Technical Contributions. In this paper, we focus on understanding the *edge privacy* risk and the strength of the privacy protection mechanisms (*e.g.*, DP) for vertically partitioned graph learning. Specifically, we make following contributions.

1) We propose the *first* query based edge re-identification attack LINKTELLER against GNN models by considering the influence propagation in GNN training. We show that it is possible to re-identify private edges effectively in a vertically partitioned graph learning setting.
2) We explore and evaluate the proposed LINKTELLER attack against different DP GCN mechanisms as countermeasures. Since there is no DP GCN mechanism proposed so far, we evaluate LINKTELLER against a standard DP strategy EDGERAND on graph, and a proposed DP GCN approach LAPGRAPH.
3) We provide formal privacy analysis for the two DP GCN approaches and an upper bound for general edge re-identification attack success rate on DP GCN mechanisms.
4) We design extensive experiments on eight datasets under the inductive setting and three datasets under the transductive setting to show that the proposed LINKTELLER is able to achieve high attack precision and recall, and significantly outperforms the random attack and two state of the art methods. We show that both DP GCN approaches are not always resilient against LINKTELLER empirically under mild privacy guarantees.
5) We systematically depict the empirical tradeoff space between (1) *model utility*—the quality of the trained GCN model, and (2) *privacy vulnerability*—the risk of a GCN model being successfully attacked. We carefully analyze different regimes under which a data holder might want to take different actions via evaluating a range of privacy budgets, and we also analyze such tradeoff by selecting a privacy budget via a validation dataset.

II. PRELIMINARIES

A. Graph Neural Networks

Graph Neural Networks (GNNs) [12] are commonly used in semi-supervised node classification tasks on graphs. Given a graph $G = (V, E)$ with V denoting the nodes ($n = |V|$) and E the edges, the adjacency matrix $A \subseteq \{0,1\}^{n\times n}$ is a sparse matrix, where $A_{ij} = 1$ denotes the existence of an edge from node i to node j. Since Graph Convolutional Network (GCN) [13] is one most representative class of GNN, we next introduce GCN, which is a stack of multiple graph convolutional layers as defined below:

$$H^{l+1} = \sigma(\widehat{A}H^l W^l), \tag{1}$$

where $\widehat{A}$ is the normalized adjacency matrix derived using a certain normalization technique and σ is the activation function. For the l-th graph convolutional layer, we denote the input node embeddings by H^l, the output by H^{l+1}, and the learnable weight by W^l. Each graph convolutional layer constructs the embeddings for each node by aggregating the embeddings of the node's neighbors from the previous layer. Specifically, H^0 is the node feature matrix X.

GNNs were first proposed for transductive training where training and testing occur on the same graph. Recently, inductive learning has been widely studied and applied [14]–[17], which is a setting where the trained GNNs are tested on unseen nodes/graphs. There are two main application scenarios for the inductive setting: 1) training on an evolving graph (*e.g.*, social networks, citation networks) for future use when more nodes arise in the graph; 2) training on one graph belonging to a group of similarly structured graphs, and transfer the model to other graphs from the similar distribution. We consider both the inductive and transductive settings in this paper.

B. Differential Privacy

Differential privacy [18] is a privacy notion that ensures an algorithm only outputs general information about its training data without revealing the information of individual records.

Definition 1 (Differential Privacy)**.** A randomized algorithm $\mathcal{M}$ with domain $\mathbb{N}^{|\mathcal{X}|}$ is $(\varepsilon, 0)$-differentially private if for all $\mathcal{S} \subseteq \text{Range}(\mathcal{M})$ and for all $x, y \in \mathbb{N}^{|\mathcal{X}|}$ such that $\|x-y\|_1 \leq 1$:

$$\Pr[\mathcal{M}(x) \in \mathcal{S}] \leq \exp(\varepsilon) \Pr[\mathcal{M}(y) \in \mathcal{S}]$$

There are two extensions of differential privacy to preserve private information in graph data. Edge differential privacy [19] protects the edge information, while node differential privacy [20] protects the existence of nodes. A recent work has proposed an algorithm to generate synthetic graphs under edge local differential privacy [21], which provides privacy protection when the graph data is distributed among different users. We consider a practical privacy model in the data partitioning scenario where one data holder only owns either the edge or node information, and we aim to protect the edge information from being leaked during the training and inference processes.

III. LinkTeller: Link Re-identification Attack

In this section, we focus on understanding the risk of edge privacy leakage caused by exposing a GNN trained on private graph data via an inference API. We first describe the **interaction model** between data holders, and then the LinkTeller algorithm that probes the *inference values* between pairs of nodes and uses these values as our confidence on whether edges exist between pairs of nodes. As we will see, this attack allows us to recover a significant number of edges from the private graph structured data.

A. Interaction Model between Data Holders

We consider an ML application based on graph structured data, where different data holders have access to different information of the graphs (*e.g.*, nodes or edges). More specifically, the graph edge information is not available to everyone,

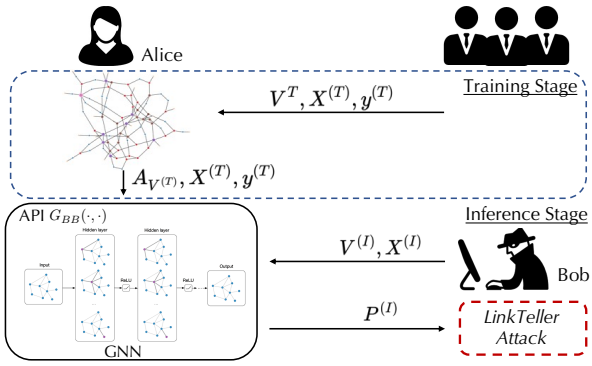


Fig. 2: **The interaction model.** In the training stage, first, some users send Alice the node set $V^{(T)}$, the associated features $X^{(T)}$ and labels $y^{(T)}$. Next, Alice trains a GNN model with corresponding adjacency matrix $A_{V^{(T)}}$, $X^{(T)}$, and $y^{(T)}$. In the inference stage, an adversarial user Bob queries Alice with a test node set $V^{(I)}$ and associated features $X^{(I)}$. Alice outputs the prediction matrix $P^{(I)}$.

since the edge connections or interactions between the node entities usually contain sensitive information, which can be exploited for malicious purposes. Thus, we first make the following abstraction of the data holder interaction.

As shown in Figure 2, the data holder Alice holds private edge information of a graph, while other users hold the node information, and due to privacy concerns, the sensitive edge information from Alice cannot be directly shared. During the *training stage*, in order to jointly train a GNN model on the graph data, some users will first send the node features of the training graph $X^{(T)}$ together with their labels $y^{(T)}$ for a set of nodes $V^{(T)}$ to Alice; and Alice will train a GNN model together with her edge connection information for future inference purpose by releasing an inference API to external users. During the *inference stage*, a potential adversarial user Bob will collect the node features of the inference graph $X^{(I)}$ (*e.g.*, patients in the next month) and obtain their predicted labels from Alice via the inference API. Alice will then send the prediction matrix $P^{(I)}$ formed by the prediction vector for each node to Bob. Without loss of generality, in the following we will use "Bob" to denote both the general users during training and inference time and the adversary during inference, although they are usually independent users in practice.

Next, we will define such training interaction formally. In particular, we will use the lower case letter to denote a vector and the upper case letter to denote a matrix. We denote the set of nodes of the training graph held by Bob as $V^{(T)} = \{v_1^{(T)}, v_2^{(T)}, \ldots, v_n^{(T)}\} \subseteq V$. In the training stage, data holder Bob will first send the corresponding node features $X^{(T)}$ and labels $y^{(T)}$ to Alice. Here each label $y_i^{(T)}$ takes the value from a set of c classes. Then the data holder Alice who holds the node connection information will generate the adjacency matrix $A_{V^{(T)}} \subseteq \{0,1\}^{n \times n}$, where $A_{V^{(T)}ij} = 1$ if and only if there is one edge between node $v_i^{(T)}$ and $v_j^{(T)}$. This way, Alice can leverage the node features and labels from Bob together with her adjacency information to train a GNN model and provide the model as a **blackbox** API, $G_{BB}(\cdot,\cdot)$, for Bob.

The learned model parameters are denoted as $\{W^i\}$, where $W^i \in \mathbb{R}^{d_i \times d_{i+1}}$ represents the weight of the i-th layer.

During the inference stage, the data holder Bob who owns another set of nodes from the inference graph will query the inference API for node prediction. In particular, given a set of inference nodes $V^{(I)} \subseteq V$, Bob will send the associated node features $X^{(I)}$ to the trained GNN API $G_{BB}(\cdot, \cdot)$. Then together with the private adjacency matrix of inference graph $A_{V^{(I)}}$, the API from Alice will make inference on the nodes, and following the standard commercial ML inference services such as Clarifai [22] and Google Vision API [23], Alice will send the logits information back to Bob as below.

$$G_{BB}(V^{(I)}, X^{(I)}) = \text{GNN}(A_{V^{(I)}}, X^{(I)}, \{W^i\}).$$

For ease of reference, we denote the output *prediction matrix* of $G_{BB}(V^{(I)}, X^{(I)})$ as $P^{(I)}$, which is of shape $|V^{(I)}| \times c$. Each row of the prediction matrix corresponds to one node in $V^{(I)}$, and each column corresponds to the confidence for one class. Alice will then send $P^{(I)}$ back to Bob.

We discuss more on the properties of $V^{(T)}$, $V^{(I)}$, and V. It is worth noting that V may not necessarily be a fixed set. New nodes and edges may arise with time elapsing, though Alice always has an up-to-date view of the graph structure. In this case, $V^{(T)}$ can be nodes in the stale graph, and $V^{(I)}$ can be the newly arisen nodes. There is also no restriction that all nodes in V should form a connected component. Rather, V can contain nodes in a group of graphs, as long as the grouping makes logical and practical sense. Under this setting, $V^{(T)}$ and $V^{(I)}$ can be the nodes of different graphs in the group.

B. Overview of the Attack

We will first introduce the capability/knowledge of the attacker, and then provide overview of the attack method. During the attack, the attacker has access to a set of node features and their labels which are required during training. During inference, Bob is able to query the trained API for multiple times with the subset of nodes that are of interest. That is to say, the attacker's **capability** includes the query access to a blackbox GNN model and the obtained prediction probability for a set of nodes during inference. Note that the attacker has no information about the API model except that it is a GNN model with unknown architecture and parameters. Unlike He *et al.* [11] which assumes the knowledge of partial graphs or a shadow dataset, here, we have no such additional assumptions.

The **overview** of the proposed link re-identification attack LINKTELLER is as follows. The attacker plays the role of Bob in the interaction model (Figure 2). The *goal* of the attacker is to recover the connections among the inference node entities. Concretely, during inference, attacker Bob will query the GNN API with a set of inference nodes. With the returned prediction probability vectors, Bob will infer the connections between certain pairs of nodes. The attack succeeds if the attacker can correctly determine whether two given nodes are connected by a link or not. We use the standard **metrics** *precision*, indicating what fraction of pairs inferred to be connected are indeed connected in the graph; and *recall*, indicating what fraction of the connected pairs are revealed, to measure the attack success rate. We also evaluate the *AUC scores*.

Intuitions: Our attack is inspired by the intuition behind the training of a GNN model—during training, if two nodes u and v are connected by an edge, GNN would "propagate" the information of u to v. As a result, if there is an edge from u to v, we would expect that changing the feature vector of u would impact the prediction of v. Thus, if we can compute the *influence* of one node on the other, we could use it to guess whether there is an edge between the two nodes: If the *influence value* is "large", we would be more confident on the existence of an edge; if the *influence value* is "small", we would be more confident that the nodes are not directly connected. Below, we describe a concrete algorithm to approximate such an influence value by probing the trained GNN inference API.

C. LINKTELLER: Edge Influence Based Attack

LINKTELLER attack proceeds in two phases. First, given a collection of nodes $V^{(I)}$ at the inference time and the inference API, $G_{BB}(\cdot, \cdot)$, the attacker tries to calcluate the *influence value* between each pair of nodes in $V^{(I)}$. Second, LINKTELLER then sorts all pairs of nodes by their influence value, and predict the first $m = \hat{k} \cdot \frac{n(n-1)}{2}$ node pairs with highest influence values as "`with edge`" and all other pairs as "`without edge`". Here $\hat{k}$ is a hyperparameter specified by the attacker, which indicates his prior "belief" of the graph density. We call $\hat{k}$ the *density belief*, which is a key hyper-parameter (details in Algorithm 1). In our experiments, we observe that the attack performance will decrease slightly given the discrepancy between estimated $\hat{k}$ and ground truth k. Nevertheless, we show that LINKTELLER remains more effective than the state-of-the-art attacks even with inaccurate estimation for $\hat{k}$.

In practice, it is also possible for attackers to further estimate $\hat{k}$ or the influence value *threshold* for edge re-identification with additional knowledge. For instance, if the attacker has partial graph information, she can either estimate $\hat{k}$, or directly calculate the influence values for known connected/unconnected pairs and estimate the threshold for distinguishing them. More concrete descriptions of such actionable strategies are deferred to Appendix E2, which we hope can inspire more effective attacks as interesting future work.

Measuring Influence Values via the Inference API: Here we describe the calculation of the influence value between a node pair. Recall that in the interaction model, for any inquiry involving a set of nodes and their features, the attacker Bob is given a prediction vector for each node. With the hope of taking advantage of the prediction vectors to obtain the influence value, we look into the structure of graph convolutional layers and analyze the influence of an edge on its incident nodes.

We characterize the influence of one node v to the other node u by measuring the change in the prediction for node u when the features of the node v get reweighted. Formally, let $V^{(I)}$ be the set of nodes involved in the inference stage and $X = [x_1^\top, \ldots, x_v^\top, \ldots]^\top$ be the corresponding feature matrix. By upweighting the features of the node v by a small value Δ, the attacker generates a new feature matrix

$X' = [x_1^\top, \ldots, (1+\Delta)x_v^\top, \ldots]^\top$. The difference between the two predictions $P^{(I)}$ and $P'^{(I)}$ with respect to Δ denotes the influence of reweighting v on the prediction of all other nodes. We define the influence matrix of v on other nodes as $I_v = \lim_{\Delta \to 0} \left(P'^{(I)} - P^{(I)}\right)/\Delta$ with size $|V^{(I)}| \times c$ Its u-th row $i_{vu} \in I_v$ represents the prediction influence vector of v on u for each class dimension. Finally, we compute the ℓ_2 norm of the corresponding influence vector as the influence value of v on u as $\|i_{vu}\|$. Since computing the influence matrix I_v yields the influence of one node v on *all* other nodes, to compute the influence value between n^2 pairs of nodes (n is the number of nodes of interest), we only need to compute n influence matrices, each for one node in the interested node set. This requires $2n$ forward passes of the trained network in all, which does not constitute a significant overhead during inference time. We report the running time of LINKTELLER in Appendix G3.

Next, we will theoretically show that the influence value $\|i_{vu}\|$ comes with a nice property for GCN models, that is, *two nodes that are at least $k+1$ hops away have no influence on each other in a k-layer graph convolutional network*. We start from the simple case of a 1-layer GCN in Proposition 1.

Proposition 1 (Influence value of a 1-layer GCN)**.** *For a 1-layer trained GCN model with parameters W, when its input adjacency matrix is A and feature matrix is X, when there is no edge between node u and node v, the influence value $\|i_{vu}\| = 0$.*

We omit the proof in Appendix A1 and next present a natural extension of the above conclusion for a k-layer GCN.

Theorem 1 (Influence value for a k-layer GCN)**.** *For a k-layer trained GCN model, when node u and node v are at least $k+1$ hops away, the influence value $\|i_{vu}\| = 0$.*

The complete proof is provided in Appendix A2. With the guarantee provided by Proposition 1, we can identify the connected pairs against a 1-layer GCN with high confidence; the criterion is that the pair is connected if and only if the influence value is non-zero. For GCNs with more layers, though this criterion does not directly apply, Theorem 1 can help to rule out nodes that are $k+1$ hops apart, thus eliminating a significant number of negative examples (*i.e.*, unconnected pairs). Moreover, the node pairs that are directly connected would have higher influence values, observed by studies on local neighborhood properties [17]. Although there is no strict guarantee that the influence values of the connected pairs are the largest since the values also depend on the features and the learned weights, in practice, the learned weights will generally display a preference for connected pairs for better label propagation, and thus the corresponding influence values of connected pairs are larger.

IV. COUNTERMEASURES OF LINKTELLER: DIFFERENTIALLY PRIVATE GCN

In this section, we aim to evaluate to what extent the proposed LINKTELLER in Section III-B reveals the private connection information effectively through a trained GCN, as well as the sufficient conditions of the attack, via considering different countermeasure approaches. The most direct countermeasure or defense against such an attack would be a differentially private GCN model. However, so far there is no existing work directly training differentially private GCN models to our best knowledge. As a result, we first revisit the **general framework** and principles of developing a differentially private (DP) GCN against such edge re-identification attacks (Section IV-A). We then formally define the DP GCN, followed by two proposed **practical algorithms** to train a DP GCN. We also discuss the upper bound of the precision of general edge re-identification attacks on DP GCNs. Importantly, we point out that the theoretical guarantee of differential privacy is insufficient in preserving both privacy and utility for a GCN. It is equally important to empirically choose an appropriate privacy budget to strike a better privacy-utility balance.

Algorithm 1: Link Re-identification Attack (LINKTELLER)

Input: A set of nodes of interest $V^{(C)} \subseteq V^{(I)}$; the associated node features X; the inference API $G_{BB}(\cdot,\cdot)$; density belief $\hat{k}$, reweighting scale Δ

Output: a $0/1$ value for each pair of nodes, indicating the absence/presence of edge

Function `InfluenceMatrix` $(V^{(I)}, X, G_{BB}(\cdot,\cdot), v)$ **:**
  $P = G_{BB}(V^{(I)}, X)$
  $X' = [x_1^\top, \ldots, (1+\Delta)x_v^\top, \ldots]^\top$
  $P' = G_{BB}(V^{(I)}, X')$
  $I = \frac{1}{\Delta}(P' - P)$
  return I

for *each node* $v \in V^{(C)}$ **do**
  $I \leftarrow$ `InfluenceMatrix` $(V^{(I)}, X, G_{BB}(\cdot,\cdot), v)$
  for *each node* $u \in V^{(C)}$ **do**
    $i_{uv} \leftarrow \|I[u,:]\|$ ▷ `The norm of the` u`-th row of` I

Sort all i_{uv} in a descending order
$n \leftarrow |V|$
$m \leftarrow \hat{k} \cdot \frac{n(n-1)}{2}$
Assign 1 to the first m pairs, and 0 to the remaining

A. Overview of DP GCN Framework

In the following sections, we review the definition of edge differential privacy for graph algorithms [19] and present two practical DP GCN training algorithms via graph structure input perturbation. Input perturbation for GCN is a non-trivial problem since naively adding noise to the graph structure would destroy the sparsity of the adjacency matrix. The loss of sparsity greatly limits a GCN's performance and increases its memory and computation cost.

To preserve the sparsity of the adjacency matrix, we discuss two approaches for GCN input perturbation: EDGERAND and LAPGRAPH. EDGERAND adapts the idea in Mülle *et al.* [24] and randomly flips each entry in the adjacency matrix according to a Bernoulli random variable. LAPGRAPH improves upon EDGERAND by pre-calculating the original graph density using a small privacy budget and using that density to clip the perturbed adjacency matrix. Compared to EDGERAND, LAPGRAPH preserves the sparsity of the adjacency matrix under a small privacy budget.

Differentially Private GCN: In edge differential privacy [19], two undirected graphs are said to be neighbors if one graph can be obtained by the other by adding/removing one edge. Definition 2 defines the neighboring relation using the adjacency matrix representation.

Definition 2 (Neighboring relation)**.** Let $\mathcal{A}$ be the set of adjacency matrices of undirected graphs. Any pair of two *symmetric* matrices $A, A' \in \mathcal{A}$ are said to be neighbors when the graph represented by A' could be obtained by adding/removing one edge from graph A, denoted as $A \sim A'$. Further, we denote the differing edge as $e = A \oplus A'$.

Definition 3 (ε-edge differential privacy)**.** A mechanism $\mathcal{M}$ is ε-edge differentially private if for all valid matrix $A \in \mathcal{A}$, and $A' \sim A$, and any subset of outputs $\mathcal{S} \subseteq Range(\mathcal{M})$, the following holds:

$$\Pr[\mathcal{M}(A) \in \mathcal{S}] \leq \exp(\varepsilon) \cdot \Pr[\mathcal{M}(A') \in \mathcal{S}]. \tag{2}$$

The probability $\Pr$ is taken over the randomness of $\mathcal{M}$.

Definition 3 formally presents the definition of ε-edge differential privacy (ε-edge DP). It guarantees that the outputs of a mechanism $\mathcal{M}$ should be indistinguishable on any pair of neighboring input graphs differing in one edge.

Next, we apply ε-edge DP to a GCN. To protect against link re-identification attacks, we need to guarantee that a GCN's inference results do not reveal the edge information of its input graph. Specifically, in the inductive training, the edge information of both the training graph and the inference graph should be protected. In addition, the privacy protection should hold even if the attacker submits an infinite number of inference queries.

Based on the above criteria, we first perturb the graphs before training a GCN. Since the training and the inference steps are both post-processing on the perturbed DP graphs, the edge information is protected for both the training and inference graphs. Moreover, submitting more queries would not reveal sensitive edge information.

We present the detailed algorithm of *perturbation*, *training*, and *inference* in a differentially private GCN framework in Algorithm 2 in Appendix D1. First, the adjacency matrix A is perturbed to meet the DP guarantee. Second, a DP GCN model is trained on a subset of training nodes in the perturbed graph. Finally, during inference, the DP GCN model is used to predict the labels for a subset of inference nodes in the perturbed graph. Essentially, ε-edge DP is achieved in the step of adjacency matrix perturbation (line 1-2), and the guarantee is provided by the privacy guarantee of the perturbation mechanism. Since the same perturbed graph is used for both the training and inference steps, making multiple inferences would not consume additional privacy budget.

The following theorem provides differential privacy guarantee for the training procedure in Algorithm 2 for GCN models.

Theorem 2 (ε-edge differentially private GCN)**.** *The DP GCN model trained by Algorithm 2 is ε-edge differentially private if the perturbation mechanism M_ε is ε-edge differentially private.*

We omit all proofs for the DP guarantees in Appendix B. Next, we show that the `Inference` step in Algorithm 2 guarantees ε-edge DP for edges in both the training and testing graph ($A_{V^{(T)}}$ and $A_{V^{(I)}}$). To prove this privacy guarantee, we first introduce the parallel composition property of ε-edge DP.

Lemma 1 (Parallel composition of ε-edge DP)**.** *If the perturbation mechanism M_ε is ε-edge differentially private and $A_1, A_2, \ldots, A_m$ are adjacency matrices with non-overlapping edges, the combination of $M_\varepsilon(A_1), M_\varepsilon(A_2), \ldots, M_\varepsilon(A_m)$ is also ε-edge differentially private.*

The following theorem guarantees differential privacy for any inference using the DP GCN model.

Theorem 3 (ε-edge differentially private GCN inference)**.** *The `Inference` step in Algorithm 2 is ε-edge differentially private for any $V^{(I)} \subseteq V$.*

The above analysis of the general DP GCN framework provides privacy guarantees for GCN models trained following the principles in Algorithm 2. Next, we will introduce two such concrete training mechanisms.

B. Practical DP GCN

In Algorithm 2, the perturbation step M_ε takes the adjacency matrix of the input graph and adds noise to the adjacency matrix to guarantee ε-edge DP. In this section, we present two practical DP mechanisms for this process.

The intuition behind perturbing the adjacency matrix is to add enough noise in the adjacency matrix to guarantee the indistinguishability between any pair of neighboring adjacency matrices A and A'—the ratio of the probability of getting the same perturbed matrix from A and A' should be bounded by a small constant ε. The smaller ε is, the stronger the protection is.

In addition to the privacy requirements, the perturbed adjacency matrix A' also needs to satisfy the following two requirements in order to be used as a training/inference graph for DP GCN. First, for large graphs, A' needs to preserve a reasonable level of sparsity to avoid huge memory consumption when training a GCN model. Second, each row in the perturbed adjacency matrix A' should represent the same node as its corresponding row in the original adjacency matrix A. This requirement ensures that the node features and labels can be associated with the right graph structure information in the perturbed adjacency matrix during training and inference.

However, the second requirement is often not satisfied by prior work on DP synthetic graph generation [21], [25]–[28]. This line of work aims at generating graphs that share similar statistics with the original graphs. Though the desired statistics of the graphs are preserved, the nodes in the generated graph and the original graph are intrinsically unrelated. Therefore, the new DP graph structure cannot be connected with the node features and labels to train a DP GCN model. More discussions on prior works are provided in the related work section.

To satisfy the privacy and utility requirements for DP GCN, we introduce two perturbation methods that directly add noise to the adjacency matrix.

*1) Edge Randomization (*EDGERAND*):* We set out with a discrete perturbation method proposed in Mülle *et al.* [24]. This algorithm was originally proposed as a pre-processing step for DP node clustering. Since the algorithm naturally preserves the sparse structure of the adjacency matrix, we adopt it as the input perturbation algorithm for DP GCN and name it EDGERAND. We present the algorithm for EDGERAND in Algorithm 3 in Appendix D2. We first randomly choose the cells to perturb and then randomly choose the target value from $\{0, 1\}$ for each cell to be perturbed.

In EDGERAND, the level of the sparsity of the perturbed adjacency matrix is purely determined by the sampling parameter s, which can be conveniently controlled to adapt to the given privacy budget ε. The relationship between s and ε is characterized in Theorem 4.

Theorem 4. EDGERAND *guarantees ε-edge DP for $\varepsilon \geq \ln\left(\frac{2}{s} - 1\right)$, $s \in (0, 1]$.*

EDGERAND guarantees differential privacy for the perturbed adjacency matrix. However, the privacy protection comes at the cost of changing the density of the perturbed graph. Let the density of the input graph to EDGERAND be k, the expectation of the density of the output graph is $k' = (1-s)k + s/2$. Take $\varepsilon = 1$ as an example, in this case, s shall be at least 0.5379 according to Theorem 4, and k' is therefore larger than $1/4$. As such, when ε is small, the perturbed graph generated by EDGERAND could have a much higher density compared to the original one. This would increase the memory consumption for training DP GCNs on large graphs and may cause memory errors when the perturbed adjacency matrix becomes too dense to fit into the memory.

*2) Laplace Mechanism for Graphs (*LAPGRAPH*):* EDGERAND is not applicable to large graphs under small privacy budgets due to the huge memory consumption caused by a dense adjacency matrix. Therefore, we propose LAPGRAPH to address this problem.

The classical idea of adding Laplace noise to the private value is also applicable to our scenario. The difference is that, in traditional scenarios, Laplace noise is applied to entities such as a database entry, while in our case, the private entity is the adjacency matrix. Therefore, additional care shall be taken to tailor the Laplace mechanism to the graph scenario.

By the definition of Laplace mechanism [18], adding a certain amount of noise to each cell in the adjacency matrix will lead to any two neighboring adjacency matrices being indistinguishable. However, directly applying this mechanism will add a huge amount of continuous noise to each cell of the adjacency matrix, which inevitably undermines the sparse property of the matrix. The loss of sparse property introduces two problems: First, it drastically increases the computation and memory cost of training a GCN. Second, adding the continuous noise in the adjacency matrix is equivalent to adding new weighted edges between almost *every* pair of nodes in the graph, which greatly impairs the utility of the adjacency matrix and, consequently, the GCN trained on it.

To retain the sparsity, after adding noise, we only keep the largest T cells as existing edges in the perturbed graph. To preserve the original graph structure, we set T to be the approximation of the number of edges in the original graph using a small portion of the differential privacy budget. We name the perturbation method LAPGRAPH and present the details in Algorithm 4 in Appendix D2. The privacy guarantee for this method is given in Theorem 5.

Compared with EDGERAND, LAPGRAPH has the advantage of better preserving the density of the original graph, especially for large graphs and small ε. Since the number of edges in a large graph is often orders of magnitude higher than the sensitivity of adding/removing a single edge, it is possible to estimate T even under a very limited privacy budget. Thus, the density of the perturbed graph is much closer to the original one than EDGERAND. This improvement makes it possible to train DP GCN on large graphs under small privacy budgets without causing memory errors.

Theorem 5. LAPGRAPH *guarantees ε-edge DP.*

Due to the lack of DP GCN approaches, here we focus on the existing technique EDGERAND and the proposed LAPGRAPH to provide DP guarantees for GCN as countermeasures to further evaluate the proposed attack LINKTELLER. We have provided the formal analysis for the privacy guarantees for EDGERAND and LAPGRAPH above, and next, we will discuss a general upper bound of edge privacy on DP GCN models.

C. Discussion: Upper Bound of Edge Re-Identification Attack Performance on DP GCN

As implied by ε-edge DP in Definition 3, it is generally difficult to tell, among the two neighboring adjacency matrices A and A', which one leads to the observed prediction. The direct consequence of the indistinguishability is that the existence of the differing edge $e = A \oplus A'$ cannot be inferred. In this section, we aim to analyze the upper bound of edge re-identification attacks against DP GCN.

Same as the attack model introduced in Section III-B, we assume the attacker has access to a set of node features and their labels without any knowledge about the GCN structure and parameters.

To start with, we formalize the link re-identification attack proposed in Section III-B as the following game between the graph owner Alice and the attacker Bob:

1) Let V be a set of nodes and $\mathcal{A}_V$ be the set of all possible adjacency matrices for graphs with nodes V. First, Alice selects an adjacency matrix $A \in \mathcal{A}_V$ uniformly at random and uses it to generate a graph.
2) Bob selects a set of training nodes $V^{(T)} \subseteq V$. He sends $V^{(T)}$ with the features and labels of $V^{(T)}$ to Alice.
3) Alice then trains an ε-edge differentially private GCN model and exposes the inference API G_{BB} to Bob.
4) Bob selects a set of inference nodes $V^{(I)} \subseteq V$ and nodes of interests $V^{(C)} \subseteq V^{(I)}$. Let $k^{(C)}$ denote the graph density

over $V^{(C)}$. For each pair of nodes $< u, v > \in V^{(C)} \times V^{(C)}$, Bob launches a link re-identification attack $\mathcal{R}_{G_{BB}}(u, v)$ to infer whether an edge exists between nodes u and v, and $\mathcal{R}_{G_{BB}}(u, v) \in \{0, 1\}$.

To obtain an upper bound for the above attack, we assume the attacker knows the inference node density $k^{(C)}$. Formally, we bound the expected precision of the link re-identification attack $\mathcal{R}$ by the following theorem.

Theorem 6. *The precision of $\mathcal{R}_{G_{BB}}$ over nodes of interests $V^{(C)}$ with density $k^{(C)}$ is upper-bounded by:*

$$\Pr_{<u,v> \in V^{(C)} \times V^{(C)}} [A_{uv} = 1 \mid \mathcal{R}_{G_{BB}}(u, v) = 1] \leq \exp(\varepsilon) \cdot k^{(C)},$$

where the probability is calculated over the randomness in the graph selection, the noise introduced by the DP GCN training, and the selection of node pair $< u, v >$.

Proof Sketch. Based on Definition 3 and Bayes' theorem, the ratio between the posterior probability $\Pr[A_{uv} = 1 \mid G_{BB} \in S]$ and the prior belief on $\Pr[A_{uv} = 1]$ is bounded by $\exp(\varepsilon)$. Since the precision of a random guess based on the prior probability (*i.e.*, the graph density) is at most $k^{(C)}$, the upper bound for the precision of a link re-identification attack on an ε-edge differentially private GCN is $\exp(\varepsilon) \cdot k^{(C)}$. The complete proof is provided in Appendix C.

Although Theorem 6 provides a theoretical upper bound for the precision of an edge re-identification attack, it may not be sufficiently tight to provide the best privacy-utility trade-off. For example, given a graph with 1% density, the attack precision is bounded below 2% (*i.e.*, no more than two times higher than random guessing using the prior probability) if and only if $\varepsilon \leq \ln 2$. However, in practice, the same empirical protection might be achieved by a model with weaker privacy protection (*i.e.*, higher privacy budget) and therefore better utility. Thus, in Section VI-B, we empirically evaluate the privacy-utility trade-off of DP GCN across multiple datasets.

In addition to DP GCN approaches, it may also be possible to leverage some heuristics to detect such attacks. For instance, one may distinguish the abnormal behavior of querying the same set of inference nodes $V^{(I)}$ multiple times (with the node features of one node slightly altered in each query). The defender could also optimize a query limit Q which decreases the attack performance while maintaining reasonable benign query accuracy, although there is no guarantee for such detection. More discussions on the detection strategies are deferred to Appendix E1, and in this paper, we will focus on the DP GCN mechanisms with privacy guarantees.

V. Evaluation of LinkTeller

We evaluate the effectiveness of the LinkTeller attack on multiple graph datasets under various scenarios compared with three baselines. In particular, we investigate how different factors such as node degree affect the attack performance.

A. Datasets

We evaluate LinkTeller on eight datasets in the *inductive* setting and three datasets in the *transductive* setting (Appendix F1) and provide a brief description of the data below. Under the inductive setting, the first dataset is the *twitch dataset* [29] which is composed of 6 graphs as disjoint sets of nodes. Each of the graphs represents a set of people in one country; the nodes within a graph represent users in one country, and the links represent mutual friendships between users. The dimension of the features is the same across different graphs and each dimension has the same semantic meaning. Some sampled features include games they like, location, and streaming habits. The task is a binary classification task which classifies whether a streamer uses explicit language. This dataset is proposed for transfer learning, *i.e.*, applying the model learned on one graph to make inferences on the other graphs corresponding to different countries. In our evaluation, we train the GNN model on the graph twitch-ES, and transfer it to other five countries (RU, DE, FR, ENGB, PTBR). *PPI* [14] and *Flickr* [16] are another two standard datasets used in graph inductive learning setting. PPI is a dataset for multi-label classification task, which aims to categorize the function of proteins across various biological protein-protein interaction graphs. Flickr is an evolving graph for the classification task, which contains descriptions and common properties of images as node features. For both PPI and Flickr, we use the standard splits for training and testing following the previous works. Under the transductive setting, we adopt three standard datasets (Cora, Citeseer, and Pubmed). More details of the data can be found in Appendix F1.

B. Models

We mainly experiment with GCN models. The configurations/hyperparameters include the normalization techniques applied to the adjacency matrix, the number of hidden layers, the number of input units, hidden units, and output units, as well as the dropout rate. For each combination of hyperparameters, we train the network to minimize the cross-entropy loss for the intended tasks. We performed grid search to get the best set of hyperparameters on the validation set. The search space for the hyperparameters and the formulae for different normalization techniques are provided in Appendix F. To measure the performance of a GCN model, we follow previous work and use F1 score for their corresponding binary classification tasks. We leave the description of the best hyper-parameters we achieve in Appendix F5. In addition to the 2-layer GCNs evaluated in the main paper, in Appendix G2, we also experimented with the 3-layer GCNs and include a discussion about GCNs of 1 layer and more than 3 layers. We conclude that LinkTeller is a successful attack against most practical GCN models. In addition, we evaluate LinkTeller on Graph Attention Networks (GATs). The details are in Section V-F.

C. Setup of the Evaluation

In this section, we first describe the metrics we use to evaluate the attack effectiveness of LinkTeller. We then present the baseline attack methods.

1) Evaluation Metrics of the attack: We use the standard metrics: **precision** (the fraction of existing edges among the pairs recognized as true by Bob) and **recall** (the fraction of edges discovered by Bob over all existing edges among the

subset of nodes). We also compute the **F1 score** (the harmonic mean of precision and recall). The reason we adopt the metric is that our problem here (distinguishing connected pairs from unconnected ones) is an imbalanced binary classification problem where the minority (the connected pair) is at the core of concern. See Appendix F2 for more details. Additionally, for fair comparison with baselines, we follow the evaluation in He *et al.* [11] and compute the **AUC score**.

2) Baseline Attacks: We compare LINKTELLER with two baselines: random attack and LSA2 attacks in He *et al.* [11].

For the random attack, we follow the standard notion and construct a random classifier as a Bernoulli random variable with parameter p which predicts true if and only if the random variable takes the value 1 [30]. Given a set of instances where a of them are true and b are false, the precision of this classifier is $a/(a+b)$ and the recall is p. In our case, a is the number of connected pairs of nodes, while $a+b$ is the number of all pairs. Therefore, *precision* is exactly the *density* k of the subset, which we formally define as $k = 2m/(n(n-1))$, where $n = |V^{(C)}|$ is the size of the set of interest and m is the number of connections among the set $V^{(C)}$. The recall of such a random classifier will be the density belief $\hat{k}$.

We also compare LINKTELLER with the state of the art LSA2 attacks [11]. In the paper, the authors discussed several types of background knowledge including node attributes, partial graph, and a shadow dataset for attackers. Among the combinations, their *Attack-2* is closest to our scenario where the attacker has only access to the target graph's node features. We follow their best practices, computing the *correlation distance* between 1) *posteriors* given by the target model and 2) *node attributes*, referred to as LSA2-post and LSA2-attr attacks.

D. Evaluation Protocol

Think about the paparazzi who are fanatical about exploiting the connections among celebrities, or the indiscriminate criminals that are maliciously targeted at the mass mediocre majority, their targets are substantially different. Consequently, the subsets they gather for attack have diverse node degree distributions. Catering to the need of evaluating our attack against *nodes of different degree distributions*, we design the scenario as follows. We consider three types of subsets that are of potential interest to the attacker: nodes of low degree, unconstrained degree, and high degree. For each type, we randomly sample a fixed number $n^{(C)}$ of nodes to form a subset $V^{(C)}$ for evaluation. When sampling nodes of low (or high) degree, we place a threshold value d_{low} (or d_{high}) and sample from nodes whose degrees are no larger than d_{low} (or no smaller than d_{high}). The value d_{low} and d_{high} are chosen empirically based on the graph. When sampling nodes of unconstrained degree, we sample nodes from the entire test set uniformly at random.

More specifically, for all datasets, we choose $n^{(C)} = |V^{(C)}| = 500$. For twitch datasets, to form the unconstrained subset, we sample from each entire testing graph. For the low degree subset and high degree subset, the threshold d_{low} and d_{high} are set to 5 and 10, respectively. We set the d_{low} value to 10 for twitch-PTBR, since the graph is much denser with abundant connections among a small number of nodes. For PPI and Flickr graphs, the subsets for testing are sampled from the testing graphs/nodes that are not involved in training. We set d_{low} as 15 and d_{high} as 30 for these two large graphs.

We also evaluate different *density belief* $\hat{k} \in \{k/4, k/2, k, 2k, 4k\}$, where k is the true density. In the experiments, we round the density k to the closest value in its most significant bit (*e.g.*, 5.61e-5 rounded to 6e-5). As we will see, the effectiveness of LINKTELLER does not heavily depend on the exact knowledge of the density k.

E. Evaluation for LINKTELLER

We first evaluate the precision, recall, and AUC of LINKTELLER on eight datasets in the inductive setting, under 3 sampling strategies (low, unconstrained, and high degree), using 5 density beliefs ($k/4, k/2, k, 2k, 4k$), compared with different baselines. For each scenario, the reported results are averaged over 3 runs using different random seeds for node sampling.

We report the precision, recall, and AUC results on some datasets in Table I and Table II and the remaining datasets in Appendix G4 due to the space limit. We leave the results of the weak random attack baseline in Appendix G1. As a brief summary, LINKTELLER significantly outperforms the random attack baseline. We mainly focus on the comparison with LSA2 attacks [11]. We show that LINKTELLER significantly outperforms these two baselines. In Table I, LSA2-{post, attr} fail to attack in most of the scenarios, while LINKTELLER attains fairly high precision and recall. The AUC scores in Table II also demonstrate the advantage of LINKTELLER. Since the baselines LSA2-{post, attr} are only performed under transductive setting in He *et al.* [11], to demonstrate the generality of LINKTELLER, we also compare with them following the same evaluation protocol as in He *et al.* [11] on three datasets in the transductive setting. The results are reported in Appendix G5. We can see that the inductive setting is indeed more challenging: the baselines always fail to attack in the inductive setting while LINKTELLER is effective; the baselines are able to re-identify some private edges in the transductive setting, while LINKTELLER is consistently more effective.

Intuitively, the high attack effectiveness of LINKTELLER compared to baselines is because that LSA2-{post, attr} only leverage node-level information (posteriors or node attributes) to perform the edge re-identification attack. Although these node-level features can be correlated with the graph structure in some graphs, this correlation is not guaranteed, especially in the inductive setting. In comparison, LINKTELLER leverages the graph-structure information inferred from the inter-node influence in a GCN model according to Theorem 1. We defer more detailed comparison and analysis in Appendix E5.

In addition, it is clear that given an accurate estimation of the density ($\hat{k} = k$), LINKTELLER achieves very high precision and recall across different node degree distributions and datasets. It is interesting to see that even when the density estimation is inaccurate (*e.g.*, $\hat{k} \in \{k/4, k/2, 2k, 4k\}$), the attack is still effective. Concretely, when the belief is smaller ($\hat{k} = k/2$), the precision values increase in all cases; when the

TABLE I: **Attack Performance (Precision and Recall)** of LINKTELLER on different datasets, compared with two baseline methods LSA2-{post, attr} [11]. Each table corresponds to a dataset. We sample nodes of low, unconstrained, and high degrees as our targets. Groups of rows represent different *density belief* $\hat{k}$ of the attacker.

twitch-RU		low		unconstrained		high	
$\hat{k}$	Method	precision	recall	precision	recall	precision	recall
$k/4$	Ours	**100.0** ± 0.0	**33.0** ± 2.8	**95.1** ± 1.1	**26.0** ± 1.0	**98.9** ± 0.2	**18.1** ± 1.3
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.2 ± 0.3	0.0 ± 0.1
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	1.3 ± 0.5	0.4 ± 0.1	2.5 ± 1.3	0.4 ± 0.2
$k/2$	Ours	**100.0** ± 0.0	**61.3** ± 5.1	**87.9** ± 0.4	**48.1** ± 2.3	**97.1** ± 0.3	**35.6** ± 2.6
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.1 ± 0.2	0.0 ± 0.1
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	1.7 ± 0.2	0.9 ± 0.1	2.5 ± 0.5	0.9 ± 0.1
k	Ours	**78.7** ± 1.9	**92.6** ± 5.5	**71.8** ± 2.2	**78.5** ± 2.4	**89.7** ± 1.7	**65.7** ± 3.9
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.1 ± 0.1	0.0 ± 0.1
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	1.1 ± 0.1	1.2 ± 0.1	2.2 ± 0.6	1.6 ± 0.3
$2k$	Ours	**42.7** ± 3.4	**100.0** ± 0.0	**43.5** ± 1.9	**95.0** ± 0.5	**62.9** ± 4.2	**91.8** ± 1.3
	LSA2-post	0.7 ± 0.9	1.8 ± 2.5	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.1
	LSA2-attr	1.3 ± 0.9	3.2 ± 2.3	0.8 ± 0.1	1.8 ± 0.3	2.0 ± 0.3	2.8 ± 0.3
$4k$	Ours	**21.3** ± 1.7	**100.0** ± 0.0	**22.5** ± 1.1	**98.1** ± 0.6	**33.6** ± 2.5	**98.0** ± 0.4
	LSA2-post	0.3 ± 0.5	1.8 ± 2.5	0.0 ± 0.0	0.1 ± 0.1	0.0 ± 0.0	0.0 ± 0.1
	LSA2-attr	0.7 ± 0.5	3.2 ± 2.3	0.7 ± 0.1	3.0 ± 0.5	1.6 ± 0.3	4.6 ± 0.5

twitch-FR		low		unconstrained		high	
$\hat{k}$	Method	precision	recall	precision	recall	precision	recall
$k/4$	Ours	**100.0** ± 0.0	**28.3** ± 2.4	**97.2** ± 0.9	**22.7** ± 0.6	**99.4** ± 0.5	**24.1** ± 2.1
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.2 ± 0.2	0.0 ± 0.1	0.5 ± 0.2	0.1 ± 0.0
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	0.6 ± 0.4	0.1 ± 0.1	1.9 ± 0.7	0.5 ± 0.1
$k/2$	Ours	**100.0** ± 0.0	**50.0** ± 0.0	**95.0** ± 1.0	**44.3** ± 1.3	**98.3** ± 1.0	**47.7** ± 4.5
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.1 ± 0.1	0.0 ± 0.1	0.3 ± 0.1	0.1 ± 0.0
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	0.6 ± 0.2	0.3 ± 0.1	1.4 ± 0.2	0.7 ± 0.0
k	Ours	**92.5** ± 5.4	**92.5** ± 5.4	**84.1** ± 3.7	**78.2** ± 1.9	**83.2** ± 1.4	**80.6** ± 6.7
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.1	0.0 ± 0.1	0.1 ± 0.0	0.1 ± 0.0
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	0.8 ± 0.2	0.7 ± 0.2	1.4 ± 0.2	1.3 ± 0.1
$2k$	Ours	**51.1** ± 1.6	**100.0** ± 0.0	**51.3** ± 2.1	**95.3** ± 1.3	**49.1** ± 2.7	**94.8** ± 3.2
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.1	0.1 ± 0.0	0.1 ± 0.0
	LSA2-attr	1.7 ± 2.4	3.3 ± 4.7	0.8 ± 0.1	1.4 ± 0.2	1.6 ± 0.2	3.1 ± 0.3
$4k$	Ours	**25.6** ± 0.8	**100.0** ± 0.0	**26.5** ± 1.1	**98.3** ± 1.0	**25.4** ± 1.8	**97.6** ± 1.6
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.1	0.0 ± 0.0	0.1 ± 0.0
	LSA2-attr	0.8 ± 1.2	3.3 ± 4.7	1.0 ± 0.2	3.7 ± 0.8	1.5 ± 0.1	5.7 ± 0.1

PPI		low		unconstrained		high	
$\hat{k}$	Method	precision	recall	precision	recall	precision	recall
$k/4$	Ours	**100.0** ± 0.0	**26.1** ± 2.2	**99.5** ± 0.7	**25.9** ± 2.7	**99.7** ± 0.3	**21.6** ± 0.8
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1.4 ± 0.6	0.3 ± 0.1
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1.3 ± 0.7	0.3 ± 0.1
$k/2$	Ours	**100.0** ± 0.0	**47.6** ± 4.7	**99.5** ± 0.8	**51.5** ± 5.4	**99.7** ± 0.2	**43.3** ± 1.6
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.3 ± 0.4	0.1 ± 0.2	1.6 ± 0.5	0.7 ± 0.2
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.6 ± 0.3	0.3 ± 0.1
k	Ours	**98.7** ± 1.9	**89.2** ± 7.9	**89.5** ± 6.5	**91.9** ± 3.7	**98.0** ± 0.3	**85.1** ± 3.2
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.3 ± 0.2	0.3 ± 0.2	2.1 ± 0.6	1.8 ± 0.6
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	0.1 ± 0.2	0.1 ± 0.2	0.3 ± 0.2	0.3 ± 0.1
$2k$	Ours	**56.7** ± 6.6	**100.0** ± 0.0	**49.0** ± 5.4	**100.0** ± 0.0	**57.7** ± 2.3	**100.0** ± 0.0
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.3 ± 0.2	0.5 ± 0.4	2.1 ± 0.1	3.6 ± 0.3
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	0.1 ± 0.1	0.1 ± 0.2	0.2 ± 0.1	0.3 ± 0.1
$4k$	Ours	**28.3** ± 3.3	**100.0** ± 0.0	**24.5** ± 2.7	**100.0** ± 0.0	**28.8** ± 1.2	**100.0** ± 0.0
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.3 ± 0.1	1.3 ± 0.3	2.0 ± 0.0	7.0 ± 0.1
	LSA2-attr	0.3 ± 0.5	1.1 ± 1.6	0.0 ± 0.0	0.1 ± 0.2	0.1 ± 0.0	0.3 ± 0.1

Flickr		low		unconstrained		high	
$\hat{k}$	Method	precision	recall	precision	recall	precision	recall
$k/4$	Ours	**83.3** ± 23.6	**26.1** ± 5.5	**63.9** ± 30.7	**18.4** ± 9.0	**14.9** ± 3.8	**3.8** ± 1.3
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1.4 ± 2.0	0.4 ± 0.6
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.7 ± 1.0	0.2 ± 0.3
$k/2$	Ours	**63.9** ± 10.4	**38.3** ± 10.3	**60.0** ± 22.5	**29.7** ± 11.7	**19.6** ± 2.8	**9.9** ± 1.9
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1.8 ± 1.1	0.9 ± 0.6
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.4 ± 0.5	0.2 ± 0.3
k	Ours	**51.0** ± 7.0	**53.3** ± 4.7	**33.8** ± 13.3	**32.1** ± 13.3	**18.2** ± 4.5	**18.5** ± 6.1
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	2.3 ± 0.7	2.3 ± 0.9
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.3 ± 0.4	0.3 ± 0.4
$2k$	Ours	**34.5** ± 6.7	**71.1** ± 15.0	**27.3** ± 8.4	**50.3** ± 16.8	**13.3** ± 1.7	**26.8** ± 5.6
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1.6 ± 0.6	3.2 ± 1.3
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.4 ± 0.4	0.8 ± 0.9
$4k$	Ours	**21.7** ± 2.4	**86.1** ± 10.4	**19.8** ± 3.0	**71.9** ± 10.6	**9.2** ± 0.8	**37.3** ± 7.1
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1.3 ± 0.3	5.3 ± 1.2
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.3 ± 0.2	1.3 ± 0.9

TABLE II: **AUC** of LINKTELLER comparing with two baselines LSA2-{post, attr}. Each column corresponds to one dataset. Groups of rows represent sampled nodes of different degrees.

Degree	Method	Dataset						
		RU	DE	FR	ENGB	PTBR	PPI	Flickr
low	Ours	**1.00** ± 0.00	**1.00** ± 0.00	**1.00** ± 0.00	**1.00** ± 0.00	**1.00** ± 0.00	**1.00** ± 0.00	**1.00** ± 0.00
	LSA2-post	0.58 ± 0.04	0.58 ± 0.09	0.67 ± 0.06	0.56 ± 0.02	0.59 ± 0.01	0.70 ± 0.05	0.65 ± 0.09
	LSA2-attr	0.72 ± 0.03	0.77 ± 0.08	0.82 ± 0.02	0.62 ± 0.05	0.74 ± 0.00	0.48 ± 0.08	0.62 ± 0.14
uncon-strained	Ours	**1.00** ± 0.00	**1.00** ± 0.00	**0.99** ± 0.00	**1.00** ± 0.00	**1.00** ± 0.00	**1.00** ± 0.00	**1.00** ± 0.00
	LSA2-post	0.51 ± 0.00	0.52 ± 0.03	0.51 ± 0.01	0.54 ± 0.01	0.51 ± 0.01	0.64 ± 0.00	0.70 ± 0.08
	LSA2-attr	0.53 ± 0.03	0.51 ± 0.02	0.53 ± 0.01	0.61 ± 0.02	0.49 ± 0.01	0.48 ± 0.02	0.49 ± 0.04
high	Ours	**1.00** ± 0.00	**1.00** ± 0.00	**0.99** ± 0.01	**0.99** ± 0.00	**0.99** ± 0.00	**1.00** ± 0.00	**0.97** ± 0.00
	LSA2-post	0.52 ± 0.01	0.51 ± 0.01	0.52 ± 0.01	0.54 ± 0.01	0.51 ± 0.00	0.57 ± 0.00	0.69 ± 0.01
	LSA2-attr	0.46 ± 0.01	0.50 ± 0.02	0.50 ± 0.01	0.55 ± 0.01	0.46 ± 0.01	0.48 ± 0.01	0.51 ± 0.01

belief is larger ($\hat{k} = 2k$), almost all recall values are above 90% except for Flickr. Even under extremely inaccurate estimations such as $\hat{k} = k/4$ (or $\hat{k} = 4k$), the precision values (or the recall values) are mostly higher than 95%. This observation demonstrates the generality of LINKTELLER. We notice that LINKTELLER's performance on Flickr is slightly poorer than other datasets. This may be because that the trained GCN on Flickr does not achieve good performance given its highly sparse structure. This implies that the parameters of the trained network on Flickr may not capture the graph structure very well, and thus negatively influencing the attack performance.

F. Beyond GCNs: LINKTELLER *on GATs*

In this section, we aim to study the effectiveness of LINKTELLER on other GNNs. Since the rule of information propagation holds almost ubiquitously in GNNs, we hypothesize that our influence analysis based LINKTELLER can also successfully attack other types of GNNs. We directly apply Algorithm 1 on another classical model—Graph Attention Networks (GATs) [31], aiming to investigate the *transferability* of our influence analysis based attack from GCNs.

We evaluate the attack on the two large datasets PPI and Flickr introduced in Table IV. For both datasets, we train a 3-layer GAT. We leave details of the architecture and hyperparameters to Appendix F6 and report the result in Table III. Although LINKTELLER still significantly outperforms the baselines, it is less effective than that on GCNs. This is mainly due to the different structures of GCNs and GATs, which leads to different influence calculations for the two models (one related to the graph convolution and the other related to the attention mechanism). We provide more discussion on conveniently adapting LINKTELLER to other architectures in Appendix E4.

VI. EVALUATION OF DIFFERENTIALLY PRIVATE GCN

In this section, we aim to understand the capability of LINKTELLER attack by experimenting with potential ways to defend against it. In particular, we examine whether it is possible to weaken the effectiveness of LINKTELLER by ensuring the ε-edge DP guarantee of the GCN model. We further investigate the utility of the DP GCN models. In the end, we demonstrate the tradeoff between privacy and utility, which may be of interest to practitioners who wish to use DP GCNs to defend against LINKTELLER.

In particular, we aim to evaluate the attack effectiveness of LINKTELLER and the model utility on four types of models: DP GCN models derived using DP mechanisms EDGERAND

TABLE III: **Attack Performance (Precision and Recall)** of LINKTELLER on GAT.

GAT, PPI		low		unconstrained		high	
$\hat{k}$	Method	precision	recall	precision	recall	precision	recall
$k/4$	Ours	**8.3** ± 11.8	**2.1** ± 2.9	**21.2** ± 9.7	**5.8** ± 3.3	**36.0** ± 5.6	**7.8** ± 0.9
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	3.4 ± 0.5	0.7 ± 0.1
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1.3 ± 0.7	0.3 ± 0.1
$k/2$	Ours	**14.3** ± 11.7	**5.3** ± 5.3	**19.5** ± 9.3	**9.9** ± 4.4	**26.6** ± 1.4	**11.5** ± 0.1
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.3 ± 0.4	0.1 ± 0.2	3.8 ± 0.8	1.7 ± 0.4
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.6 ± 0.3	0.3 ± 0.1
k	Ours	**20.5** ± 3.6	**12.5** ± 4.5	**12.7** ± 6.4	**12.8** ± 5.8	**18.5** ± 2.1	**16.0** ± 1.7
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.4 ± 0.3	0.4 ± 0.4	3.3 ± 0.8	2.8 ± 0.6
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	0.1 ± 0.2	0.1 ± 0.2	0.3 ± 0.2	0.3 ± 0.1
$2k$	Ours	**10.7** ± 1.9	**12.5** ± 4.5	**7.5** ± 3.8	**15.1** ± 6.5	**12.5** ± 1.0	**21.7** ± 1.4
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.5 ± 0.2	1.1 ± 0.6	3.0 ± 0.4	5.3 ± 0.5
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	0.1 ± 0.1	0.1 ± 0.2	0.2 ± 0.1	0.3 ± 0.1
$4k$	Ours	**5.3** ± 0.9	**12.5** ± 4.5	**5.4** ± 1.5	**21.7** ± 3.8	**7.9** ± 0.7	**27.5** ± 1.3
	LSA2-post	0.7 ± 0.9	2.1 ± 2.9	0.9 ± 0.1	3.6 ± 0.9	2.7 ± 0.2	9.2 ± 0.5
	LSA2-attr	0.3 ± 0.5	1.1 ± 1.6	0.0 ± 0.0	0.1 ± 0.2	0.1 ± 0.0	0.3 ± 0.1

GAT, Flickr		low		unconstrained		high	
$\hat{k}$	Method	precision	recall	precision	recall	precision	recall
$k/4$	Ours	**33.3** ± 47.1	**8.3** ± 11.8	**8.3** ± 11.8	**2.4** ± 3.4	**14.5** ± 3.2	**3.6** ± 0.3
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.4 ± 0.5	0.1 ± 0.1
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.7 ± 1.0	0.2 ± 0.3
$k/2$	Ours	**16.7** ± 23.6	**8.3** ± 11.8	**4.8** ± 6.7	**2.4** ± 3.4	**7.3** ± 1.7	**3.6** ± 0.3
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.4 ± 0.3	0.2 ± 0.1
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.4 ± 0.5	0.2 ± 0.3
k	Ours	**8.3** ± 11.8	**8.3** ± 11.8	**5.9** ± 4.3	**5.7** ± 4.2	**4.2** ± 1.0	**4.2** ± 0.7
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.2 ± 0.2	0.2 ± 0.1
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.3 ± 0.4	0.3 ± 0.4
$2k$	Ours	**4.2** ± 5.9	**8.3** ± 11.8	**3.0** ± 2.2	**5.7** ± 4.2	**2.6** ± 0.6	**5.1** ± 0.9
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.4 ± 0.2	0.7 ± 0.4
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.4 ± 0.4	0.8 ± 0.9
$4k$	Ours	**2.2** ± 3.1	**8.3** ± 11.8	**1.5** ± 1.1	**5.7** ± 4.2	**1.3** ± 0.3	**5.1** ± 0.9
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.3 ± 0.0	1.2 ± 0.2
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.3 ± 0.2	1.3 ± 0.9

and LAPGRAPH, vanilla GCN models which have no privacy guarantee, as well as multi-layer perceptron (MLP) models with only node features. We note that MLP can be viewed as "perfectly" private since the edge information is not involved.

A. Datasets and Models

We use the datasets described in Section V-A. The DP GCN models are derived using DP mechanisms EDGERAND and LAPGRAPH under various privacy guarantees. For each privacy budget ε, we execute the procedure outlined in Algorithm 2, first getting a perturbed copy of the adjacency matrix, then using the perturbed training graph to train a GCN. We follow the criteria in Section V-B for parameter searching and model training, and leave more descriptions to Appendix F4.

We provide an evaluation of the model utility, aiming to characterize the tradeoff between model utility and the success rate of LINKTELLER. In evaluating the utility of the DP GCNs, we compare with two baseline models: 1) vanilla GCNs which are expected to have higher utility, though vulnerable to LINKTELLER as previously shown; 2) MLPs trained only on node features which may achieve lower classification utility but provide perfect protection of the edge information due to the non-involvement of edge information in the model.

B. DP GCN against LINKTELLER

We validate the effectiveness of LINKTELLER under various levels of privacy guarantees. We first provide the experimental setup, followed by concrete results including comparisons of the attack effectiveness of LINKTELLER on different models.

1) Experimental Setup: We inspect the effectiveness of LINKTELLER on DP GCN using the same setup as Section V-D. Similar to Section V-E, we use *precision* and *recall* to evaluate the attack. For each dataset, we consider all combinations of 2 DP mechanisms (EDGERAND and LAPGRAPH), 10 privacy budgets $(1.0, 2.0, \ldots, 10.0)$, 3 sampling strategies (low, unconstrained, and high degree), and 5 density beliefs $(k/4, k/2, k, 2k, 4k)$. The reported result of each scenario is averaged over 3 runs with different random seeds for sampling.

2) Evaluation Results: We leave the full evaluation results for all scenarios in Appendix G4 and focus on density belief $\hat{k} = k$ here. In Figure 3(b), we plot the *F1 score* of the attack w.r.t. DP budget ε. We see that the effectiveness of LINKTELLER decreases as a result of applying DP. Particularly, the F1 score becomes almost 0 when the privacy budget ε becomes smaller. When ε is large, however, the protection offered by DP is limited. In these cases, LINKTELLER is able to achieve a success rate close to that of attacking the non-private baseline.

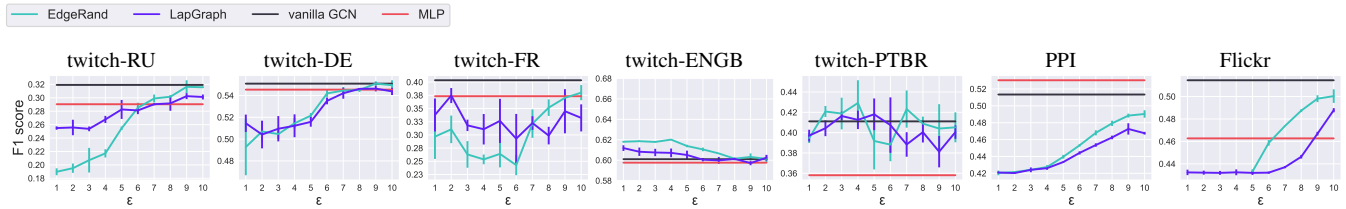
We also note that the node degree distribution has an impact on the performance of LINKTELLER. The trend is clear that as the node degree increases, the attack success rate increases substantially. Together with our previous observation in the non-private scenario that the attack success rate does not differ much for varying node degrees, we conclude that DP can offer better protection to low degree nodes than high degree nodes. We offer a simple and intuitive explanation as follows. By the design of EDGERAND and LAPGRAPH, the perturbations in all cells of the matrix are independent. As a result, nodes of low degree (those incident to fewer edges) are more susceptible to the influence, and therefore better protected by DP.

C. Model Utility Given DP Protection

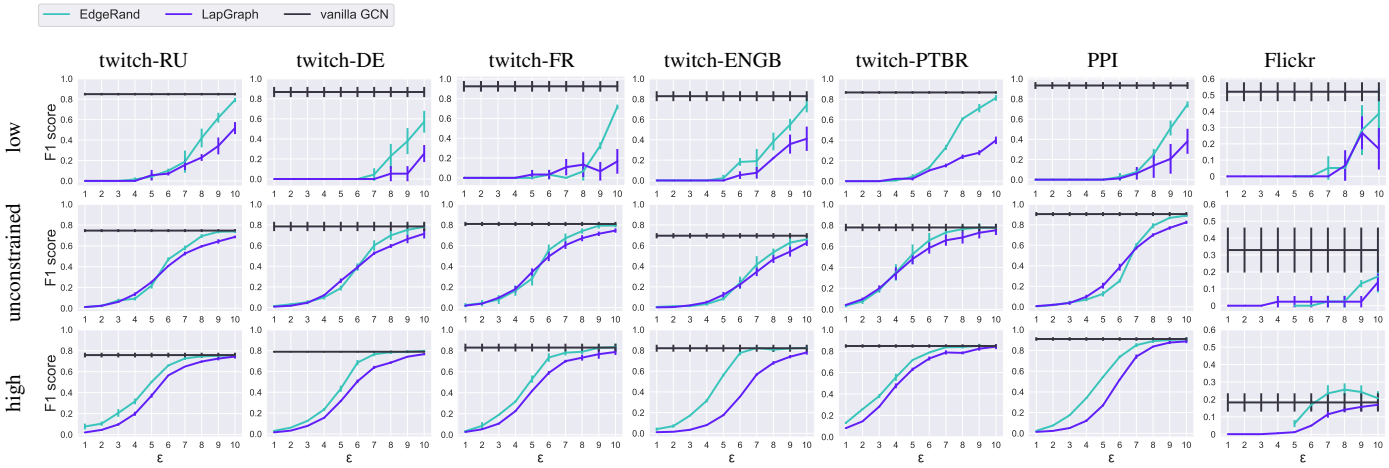
We next present the evaluation of the model utility, not only to complement the evaluation of the DP GCNs, but also to provide insights about the tradeoff between model utility and robustness against the LINKTELLER attack.

1) Experimental Setup: We evaluate the influence of applying DP (EDGERAND and LAPGRAPH) on the utility of the GCN models by comparing the results with two baseline models (a non-private vanilla GCN model and a "perfectly" private MLP baseline). We adopt the same metric to evaluate the utility of all four models: F1 score of the rare class for the twitch datasets and micro-averaged F1 score for PPI and Flickr datasets. The rationale is put in Appendix F2.

2) Evaluation Results: The figures for the model utility are presented in Figure 3(a). We plot the change of the utility with the increase of privacy budget of two DP mechanisms EDGERAND and LAPGRAPH, as well as the utility of two baseline models independent of the privacy budget. For each privacy budget ε, the reported results are averaged over 10 runs for different random seeds. We examine Figure 3(a) to see how the model utility of DP methods compares with the baselines. We first compare the performance of two baseline models: GCN (the black horizontal line) and MLP (the red line). We note that GCN is almost always better than MLP except on the PPI dataset. The observation is well suited to our intuition



(a) Model utility



(b) Attack effectiveness on different models and node degree distributions (low, unconstrained, and high)

Fig. 3: (a) Model utility and (b) attack effectiveness on different models ($\hat{k} = k$). Each column corresponds to a dataset. We consider four types of models: EDGERAND, LAPGRAPH, vanilla GCN, and MLP, with the first two satisfying DP guarantees. In each figure, the vertical bar represents the standard deviation.

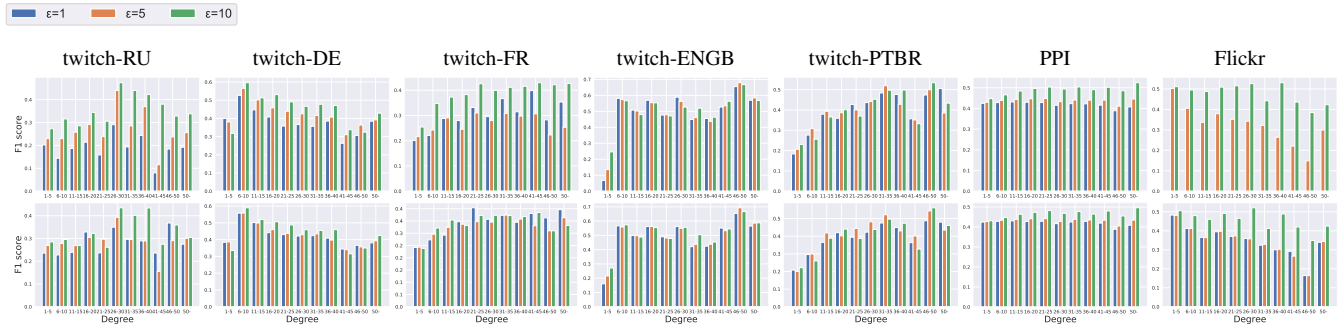


Fig. 4: F1 score of nodes with different degrees under privacy budget $\varepsilon \in \{1, 5, 10\}$ for two DP mechanisms (EDGERAND and LAPGRAPH). Each bar group represents a degree range, *e.g.*, 1-5, 6-10, 50-. Each bar within a group corresponds to one privacy budget.

that the knowledge of the graph structure can benefit learning. For most datasets, the model utility increases with the growth of the privacy budget, since the privacy protection to the graph structure becomes weaker. However, **twitch-ENGB** is an exception that achieves slightly higher utility when more privacy noise is added. This may be due to the following reason. For twitch datasets, the model is trained on the graph of twitch-ES and tested on graphs of other five countries. When the training and testing graph distribution distance is too large, there is no guarantee that the performance on the training graph can be transferred to the testing graph. This is the case for twitch-ENGB, which is an extremely sparse graph (see Table IV(a)) compared with the training graph. Thus, with slightly more random noise added, its generalization may be improved. In addition to the evaluation over a range of ε, we also evaluate such tradeoff of model utility and privacy resiliency by selecting appropriate ε based on a validation dataset. The detailed setups and results are omitted to Appendix G6.

D. Tradeoff between Model Utility and Privacy

We analyze the tradeoff between model utility and the attack performance: models with high utility tend to be more vulnerable to LINKTELLER. The sweet spot differs across datasets and scenarios. Summarizing the observations, we derive a series of conclusions on *how to protect privacy given the gap of the model utility between the vanilla GCN and the MLP model*. **First**, if the utility of the vanilla GCN is much higher than the

MLP model, then there is space for performance degradation caused by ensuring privacy. We do observe a few cases where the utility of the DP model is above the MLP baseline, and the attack success rate at that point is relatively low, especially under the low degree case, *e.g.*, the DP model on twitch-RU when $\varepsilon = 7$. In such cases, carefully choosing an ε will give the practitioner fairly good *utility* and a certain level of *privacy* guarantee simultaneously. **Second**, when the performance of the vanilla GCN only exceeds MLP by a small margin, almost all DP models that can effectively defend against the attack suffer tremendous utility loss. We point out that most scenarios fall under this category, where either privacy or utility will be sacrificed. This further substantiates the power of LINKTELLER. **Third**, when the graph structure hurts learning (*e.g.*, PPI), we may avoid using the graph structure in training by using MLP. There might exist other graph neural networks that can achieve better performance on datasets like PPI, and applying LINKTELLER to these models are exciting future work.

Utility and privacy of low-degree nodes. As noted in Section VI-B, DP GCN offers better protection to nodes of low degree. A natural question is then: *does better protection imply a degradation of utility of these nodes?* To answer this question, we separate the nodes into bins by degree (*e.g.*, 1-5, 6-10, ..., 46-50, 50-), and investigate the F1 score of nodes in each individual bin. The results on all datasets, two DP mechanisms, with three privacy budgets are presented in Figure 4. We can see that the utility for low-degree nodes does not drop faster than high-degree nodes when the privacy budget decreases, which indicates that DP GCN does not sacrifice the utility of low-degree nodes particularly.

Discussion: EDGERAND or LAPGRAPH. We further compare the results of the two mechanisms regarding model utility and attack success rate. When ε is small, the utility of EDGERAND and LAPGRAPH do not differ much (especially on PPI). When ε is large, EDGERAND generally has better model utility, while LAPGRAPH is more robust to LINKTELLER. The results for EDGERAND are incomplete for the large scale dataset Flickr under tight privacy budgets ($\varepsilon \in \{1, 2, 3, 4\}$) using EDGERAND. Under these cases, the graphs become much denser after perturbation of large magnitudes, and we experience an OOM error using an 11 GB GPU. In comparison, LAPGRAPH does not suffer such an issue.

VII. RELATED WORK

1) Privacy Attack on Graphs: This topic was widely studied [32]–[35] before graph neural networks came into play. There are mainly three types of privacy attacks on graphs: identity disclosure, attribute disclosure, and link re-identification [32], corresponding to different components (nodes, node attributes, and edges) of a graph. In this paper, we focus on *edge privacy*. Previous endeavors have illustrated the feasibility of the link re-identification attack, whilst relying on strong prior knowledge and information that arguably might not always hold or accessible. For example, when prior knowledge about the graph is available—*e.g.*, nodes with similar attributes or predictions are likely connected—He *et al.* [11] claim that an attacker could infer links in the training graph by applying methods such as clustering to predict connections for nodes within the same cluster. Duddu *et al.* [36] show that with access to the node embeddings trained to preserve the graph structure, one can recover edges by analyzing predictions based on the embeddings. Apart from the privacy attacks, there exist other adversarial attacks on GNNs, *e.g.*, against node embeddings [37] and graph- and node-level classifiers [38]. Despite the promising attacks illustrated by these early endeavors, there is a clear need to weaken the assumptions for more reliable, practical settings. In this paper, we thus answer: to what extent can we recover private edges of a graph by probing a trained blackbox GNN model *without* strong prior knowledge? Could we leverage the property of influence propagation among nodes in GNNs to design an effective attack?

2) Differential Privacy for Graphs: Differential privacy [18] is a notion of privacy that entails that the outputs of the model on neighboring inputs are close. This privacy requirement ends up obscuring the influence of any individual training instance on the model output. There are a series of works that examine the theoretical guarantee or the practical performance of models under differential privacy guarantees [39]–[44]. Depending on the properties of the datasets, *e.g.*, the distinction between the distribution of members and non-members and the underlying correlations within the datasets, there is no trivial answer to this problem.

The extension of differential privacy to the graph setting was first conducted in Hay *et al.* [45]. Since then, there has been extensive research on computing graph statistics such as degree distribution [45], cut queries [46], and sub-graph counting queries [47] under edge or node differential privacy. These statistics are useful for graph analysis but insufficient for training a GCN model. Thus, in this paper, to evaluate the strength of the LINKTELLER attack, we adapt one existing algorithm EDGERAND and propose a Laplacian mechanism LAPGRAPH for training DP GCN models as evaluation baselines.

VIII. CONCLUSIONS

We propose the first edge re-identification attack LINKTELLER via influence analysis against GNNs. We also evaluate LINKTELLER against differentially private GNNs trained using an existing and a proposed DP mechanisms EDGERAND and LAPGRAPH to understand the capability of the attack. Extensive experiments on real-world datasets (8 for inductive and 3 for transductive setting) demonstrate the effectiveness of LINKTELLER in revealing private edge information, even when there are certain privacy guarantees provided by a DP mechanism.

We believe this work will inspire a range of future research opportunities and lay down a foundation for future explorations by providing a clear data isolation problem setup, analysis of edge privacy, together with extensive empirical observations and conclusions.

Acknowledgement. This work is partially supported by the NSF grant No.1910100, NSF CNS 20-46726 CAR, NSF TRASE (ECCS-2020289), and Amazon Research Award.

REFERENCES

[1] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[2] Y. Wei, X. Wang, L. Nie, X. He, R. Hong, and T.-S. Chua, "Mmgcn: Multi-modal graph convolution network for personalized recommendation of micro-video," in *Proceedings of the 27th ACM International Conference on Multimedia*, 2019, pp. 1437–1445.

[3] J. Zhou, C. Chen, L. Zheng, X. Zheng, B. Wu, Z. Liu, and L. Wang, "Privacy-preserving graph neural network for node classification," *arXiv preprint arXiv:2005.11903*, 2020.

[4] L. Zhao, Y. Song, C. Zhang, Y. Liu, P. Wang, T. Lin, M. Deng, and H. Li, "T-gcn: A temporal graph convolutional network for traffic prediction," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 9, p. 3848–3858, Sep 2020. [Online]. Available: http://dx.doi.org/10.1109/TITS.2019.2935152

[5] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems." in *KDD*, Y. Guo and F. Farooq, Eds. ACM, 2018, pp. 974–983. [Online]. Available: http://dblp.uni-trier.de/db/conf/kdd/kdd2018.html#YingHCEHL18

[6] J. Jiang, J. Chen, T. Gu, K. R. Choo, C. Liu, M. Yu, W. Huang, and P. Mohapatra, "Anomaly detection with graph convolutional networks for insider threat and fraud detection," in *MILCOM 2019 - 2019 IEEE Military Communications Conference (MILCOM)*, 2019, pp. 109–114.

[7] "Vertex ai — google cloud," https://cloud.google.com/vertex-ai, (Accessed on 06/22/2021).

[8] "Parlai," https://ai.facebook.com/tools/parlai, (Accessed on 06/22/2021).

[9] "Infosphere virtual data pipeline — ibm," https://www.ibm.com/products/ibm-infosphere-virtual-data-pipeline, (Accessed on 06/22/2021).

[10] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," *arXiv preprint arXiv:1810.05997*, 2018.

[11] X. He, J. Jia, M. Backes, N. Z. Gong, and Y. Zhang, "Stealing links from graph neural networks," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/he-xinlei

[12] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, 2020.

[13] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," in *Proceedings of the 5th International Conference on Learning Representations*, ser. ICLR '17, 2017. [Online]. Available: https://openreview.net/forum?id=SJU4ayYgl

[14] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs." in *NIPS*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017, pp. 1024–1034. [Online]. Available: http://dblp.uni-trier.de/db/conf/nips/nips2017.html#HamiltonYL17

[15] Y. Rong, W. Huang, T. Xu, and J. Huang, "Dropedge: Towards deep graph convolutional networks on node classification." in *ICLR*. OpenReview.net, 2020. [Online]. Available: http://dblp.uni-trier.de/db/conf/iclr/iclr2020.html#RongHXH20

[16] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. K. Prasanna, "Graphsaint: Graph sampling based inductive learning method." in *ICLR*. OpenReview.net, 2020. [Online]. Available: http://dblp.uni-trier.de/db/conf/iclr/iclr2020.html#ZengZSKP20

[17] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5453–5462.

[18] C. Dwork and A. Roth, *The Algorithmic Foundations of Differential Privacy*, 2014. [Online]. Available: https://www.cis.upenn.edu/~aaroth/Papers/privacybook.pdf

[19] V. Karwa, S. Raskhodnikova, A. Smith, and G. Yaroslavtsev, "Private analysis of graph structure," *ACM Trans. Database Syst.*, vol. 39, no. 3, Oct. 2014. [Online]. Available: https://doi.org/10.1145/2611523

[20] S. P. Kasiviswanathan, K. Nissim, S. Raskhodnikova, and A. Smith, "Analyzing graphs with node differential privacy," in *Theory of Cryptography Conference*. Springer, 2013, pp. 457–476.

[21] Z. Qin, T. Yu, Y. Yang, I. Khalil, X. Xiao, and K. Ren, "Generating synthetic decentralized social graphs with local differential privacy," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 425–438.

[22] G. Sood, *clarifai: R Client for the Clarifai API*, 2017, r package version 0.4.2.

[23] "Vision ai — derive image insights via ml — cloud vision api," https://cloud.google.com/vision, (Accessed on 08/01/2021).

[24] Y. Mülle, C. Clifton, and K. Böhm, "Privacy-integrated graph clustering through differential privacy," *CEUR Workshop Proceedings*, vol. 1330, pp. 247–254, 01 2015.

[25] A. Sala, X. Zhao, C. Wilson, H. Zheng, and B. Y. Zhao, "Sharing graphs using differentially private graph models," in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, 2011, pp. 81–98.

[26] Y. Wang and X. Wu, "Preserving differential privacy in degree-correlation based graph generation," vol. 6, no. 2, p. 127–145, Aug. 2013.

[27] Q. Xiao, R. Chen, and K.-L. Tan, "Differentially private network data release via structural inference," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 911–920.

[28] S. Brunet, S. Canard, S. Gambs, and B. Olivier, "Novel differentially private mechanisms for graphs." *IACR Cryptology ePrint Archive*, vol. 2016, p. 745, 2016. [Online]. Available: http://dblp.uni-trier.de/db/journals/iacr/iacr2016.html#BrunetCGO16

[29] B. Rozemberczki, C. Allen, and R. Sarkar, "Multi-scale attributed node embedding," 2019.

[30] P. Flach and M. Kull, "Precision-recall-gain curves: Pr analysis done right," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. Curran Associates, Inc., 2015, pp. 838–846. [Online]. Available: https://proceedings.neurips.cc/paper/2015/file/33e8075e9970de0cfea955afd4644bb2-Paper.pdf

[31] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[32] E. Zheleva and L. Getoor, "Preserving the privacy of sensitive relationships in graph data," in *Proceedings of the First SIGKDD International Workshop on Privacy, Security, and Trust in KDD (PinKDD 2007)*, ser. Lecture Notes in Computer Science, vol. 4890. Springer, March 2007, pp. 153–171.

[33] M. Hay, G. Miklau, D. Jensen, P. Weis, and S. Srivastava, "Anonymizing social networks," University of Massachusetts Amherst, Tech. Rep. 07-19, March 2007.

[34] L. Zhang and W. Zhang, "Edge anonymity in social network graphs." in *CSE (4)*. IEEE Computer Society, 2009, pp. 1–8. [Online]. Available: http://dblp.uni-trier.de/db/conf/cse/cse2009-4.html#ZhangZ09

[35] A. M. Fard, K. Wang, and P. S. Yu, "Limiting link disclosure in social network analysis through subgraph-wise perturbation." in *EDBT*, E. A. Rundensteiner, V. Markl, I. Manolescu, S. Amer-Yahia, F. Naumann, and I. Ari, Eds. ACM, 2012, pp. 109–119. [Online]. Available: http://dblp.uni-trier.de/db/conf/edbt/edbt2012.html#FardWY12

[36] V. Duddu, A. Boutet, and V. Shejwalkar, "Quantifying privacy leakage in graph embedding," 2020.

[37] A. Bojchevski and S. Günnemann, "Adversarial attacks on node embeddings via graph poisoning," in *International Conference on Machine Learning*. PMLR, 2019, pp. 695–704.

[38] H. Dai, H. Li, T. Tian, X. Huang, L. Wang, J. Zhu, and L. Song, "Adversarial attack on graph structured data," in *International conference on machine learning*. PMLR, 2018, pp. 1115–1124.

[39] N. Li, W. Qardaji, D. Su, Y. Wu, and W. Yang, "Membership privacy: A unifying framework for privacy definitions," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 889–900. [Online]. Available: https://doi.org/10.1145/2508859.2516686

[40] S. Yeom, I. Giacomelli, M. Fredrikson, and S. Jha, "Privacy risk in machine learning: Analyzing the connection to overfitting," *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pp. 268–282, 2018.

[41] B. Jayaraman and D. Evans, "Evaluating differentially private machine learning in practice," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019,

pp. 1895–1912. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/jayaraman
[42] Ú. Erlingsson, I. Mironov, A. Raghunathan, and S. Song, "That which we call private," *CoRR*, vol. abs/1908.03566, 2019. [Online]. Available: http://arxiv.org/abs/1908.03566
[43] B. Jayaraman, L. Wang, K. Knipmeyer, Q. Gu, and D. Evans, "Revisiting membership inference under realistic assumptions," 2020.
[44] T. Humphries, M. Rafuse, L. Tulloch, S. Oya, I. Goldberg, and F. Kerschbaum, "Differentially private learning does not bound membership inference," 2020.
[45] M. Hay, C. Li, G. Miklau, and D. Jensen, "Accurate estimation of the degree distribution of private networks," in *2009 Ninth IEEE International Conference on Data Mining*. IEEE, 2009, pp. 169–178.
[46] J. Blocki, A. Blum, A. Datta, and O. Sheffet, "The johnson-lindenstrauss transform itself preserves differential privacy," in *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*. IEEE, 2012, pp. 410–419.
[47] ——, "Differentially private data analysis of social networks via restricted sensitivity," in *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, 2013, pp. 87–96.
[48] F. D. McSherry, "Privacy integrated queries: an extensible platform for privacy-preserving data analysis," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, 2009, pp. 19–30.
[49] W. Brendel, J. Rauber, and M. Bethge, "Decision-based adversarial attacks: Reliable attacks against black-box machine learning models," *arXiv preprint arXiv:1712.04248*, 2017.
[50] Y. Dong, H. Su, B. Wu, Z. Li, W. Liu, T. Zhang, and J. Zhu, "Efficient decision-based black-box adversarial attacks on face recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 7714–7722.
[51] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

APPENDIX

A. Proofs for Influence Analysis in LINKTELLER

1) Proof of Proposition 1:

Proof. Consider the 1-layer GCN, where

$$\text{GCN}(A, X, W) = AXW. \tag{3}$$

For the simplicity of the proof, we ignore the normalization applied to the adjacency matrix A, since it only changes the scale of numbers in the matrix. We next calculate the influence value of a pair of nodes against this 1-layer GCN according to the function `InfluenceValue` described in Algorithm 1.

Following the notation in Algorithm 1, let the set of inference nodes be $V^{(I)}$, the node feature matrix associated with the inference node set be X. We further denote the adjacency matrix induced on this node set as A. Taking a pair of nodes u, v from the set of nodes of interest $V^{(C)}$, we calculate the influence value of this pair of nodes following the steps in `InfluenceMatrix`:

$$\begin{aligned} P &= G_{BB}(V^I, X) = \text{GCN}(A, X, W) = AXW, \\ X' &= \left[x_1^\top, \ldots, (1+\Delta)x_v^\top, \ldots, \right]^\top, \\ P' &= G_{BB}(V^I, X') = \text{GCN}(A, X', W) = AX'W, \\ P' - P &= A(X' - X)W = AX_\Delta W, \end{aligned}$$

where X_Δ is a matrix of the same size as X that only contains value in each v-th row. Specifically, the v-th row vector of X_Δ is equal to Δx_v, where Δ is the reweighting coefficient.

We next compute the u-th row of the influence matrix $I = AX_\Delta W$. We start from computing $X_\Delta W$

$$X_\Delta W = \left[\mathbf{0}, \ldots, \Delta x_v^\top, \ldots, \mathbf{0}\right]^\top W = \Delta \left[\mathbf{0}, \ldots, x_v^\top, \ldots, \mathbf{0}\right]^\top W$$

Therefore, for I defined as $I = (P' - P)/\Delta = AX_\Delta W$, its u-th row is $A_{uv}x_v W$. When there is no edge between u and v, $A_{uv} = 0$, and therefore the row vector is an all zero vector. The influence value, which is the ℓ_2 norm of the row vector, is therefore 0. □

2) Proof of Theorem 1: We first present a lemma.

Lemma 2. *Let $A \in \{0,1\}^{n\times n}$ be the adjacency matrix of the graph, $H_i, H_i' \in \mathbb{R}^{n\times d_i}$ be two hidden feature matrices of size that differ in t_i rows $\{r_1, \ldots, r_{t_i}\}$ corresponding to the nodes $\{v_{r_1}, \ldots, v_{r_{t_i}}\}$, and $W^i \in \mathbb{R}^{d_i \times d_{i+1}}$ be the weight matrix of the i-th graph convolutional layer, then*

(1) AH_iW^i and $AH_i'W^i$ differ in at most t_{i+1} rows corresponding to the node set $\bigcup_{l=1}^{t_i} \mathcal{N}(v_{r_l})$, where $\mathcal{N}(u)$ denotes the neighbor set of node u.

(2) Further, let $H_{i+1} = \sigma(AH_iW^i)$ and $H_{i+1}' = \sigma(AH_i'W^i)$, then H_{i+1} and H_{i+1}' also differ in at most t_{i+1} rows corresponding to the node set $\bigcup_{l=1}^{t_i} \mathcal{N}(v_{r_l})$.

We next use Lemma 2 to help with the proof of Theorem 1.

Proof. Consider a k-layer GCN which is a stack of k graph convolutional layers defined as below

$$\text{GCN}(A, X, \{W^i\}) = A \cdots \sigma(A\sigma(AXW^1)W^2) \cdots W^k. \tag{4}$$

Its *input* feature matrices are $H_0 = X$ and $H_0' = X'$ that differ in one row; let it correspond to node u. Its *output* feature matrices are $AH_{k-1}W^{k-1}$ and $AH_{k-1}'W^{k-1}$.

Since H_0 and H_0' differ only in node u, according to Lemma 2-(2), H_1 and H_1' differ in the rows corresponding to $\mathcal{N}(u)$ (which contains nodes that are 1 hop away from u); H_2 and H_2' differ in the rows corresponding to $\bigcup_{v\in\mathcal{N}(u)} \mathcal{N}(v)$ (which contains nodes that are at most 2 hops away from u). Iteratively applying Lemma 2-(2), we see that H_{k-1} and H_{k-1}' differ in rows corresponding to nodes that are at most $k-1$ hops away from u. We finally apply Lemma 2-(1) and obtain the conclusion that $AH_{k-1}W^{k-1}$ and $AH_{k-1}'W^{k-1}$ differ in rows corresponding to nodes that are at most k hops away from u. Thus, the influence matrix

$$P_\Delta = AH_{k-1}'W^{k-1} - AH_{k-1}W^{k-1}$$

has at most t_k non-zero rows corresponding to nodes that are at most k hops away from u. It thus follows that when u and v are at least $k+1$ hops away, the v-th row of the influence matrix of node u is an all-zero row. Since the influence value of u on v is the norm of the v-th row, we thus can conclude that the *influence value* $\|i_{v_u}\| = 0$. □

Finally, we complete the proof for Lemma 2.

Proof. First of all, for H_i and H_i' that differs in t_i rows, it is obvious that H_iW^i and $H_i'W^i$ differ in the same t_i rows. For simplicity, we denote H_iW^i as F_i and $H_i'W^i$ as F_i'. We next present the condition for AF_i and AF_i' to differ. Consider the element in j-th row and k-th column of AF_i and AF_i', which are $\sum_{p=1}^n A_{jp}(F_i)_{pk}$ and $\sum_{p=1}^n A_{jp}(F_i)'_{pk}$, respectively. We first note that when $A_{jp} = 0$, the difference of $(F_i)_{pk}$ and $(F_i)'_{pk}$ does not matter. Next, for all p such that $A_{jp} = 1$, only when $(F_i)_{pk} \neq (F_i)'_{pk}$ will the difference of the product

contribute to the difference of the sum. The two points jointly imply that, if $j \notin \bigcup_{l=1}^{t_i} \mathcal{N}(v_{r_l})$, then $(F_i)_{pk} = (F_i)'_{pk}$ for all k. Thus, AF_i and AF'_i differ in at most t_{i+1} rows, corresponding to the node set $\bigcup_{l=1}^{t_i} \mathcal{N}(v_{r_l})$. Hence we establish the first claim.

For the second claim, we notice that the activation layer such as ReLU used in the standard GCN [13] is point-wise operation. Thus $\sigma(AF_i)$ and $\sigma(AF'_i)$ cannot differ in more rows, meaning H_{i+1} and H'_{i+1} will also differ at most in the rows corresponding to $\bigcup_{l=1}^{t_i} \mathcal{N}(v_{r_l})$. □

B. Proofs for Privacy Guarantees of the DP mechanisms

1) Proof of Theorem 2:

Proof. Since $\widetilde{A}_{V^{(T)}}$ is perturbed to meet ε-edge differential privacy and other inputs (*e.g.*, node features) in Algorithm 2 are independent of the graph structure, the DP GCN model is ε-edge differentially private due to the post-processing property of differential privacy. □

2) Proof of Lemma 1:

Proof. Lemma 1 extends the parallel composition property of differential privacy [48] to edge differential privacy. The parallel composition property states that if $M_1, M_2, \ldots, M_m$ are algorithms that access disjoint datasets $D_1, D_2, \ldots, D_m$ such that each M_i satisfies ε_i-differential privacy, then the combination of their outputs satisfies ε-differential privacy with $\varepsilon = \max(\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_m)$. Since the adjacency matrices $A_1, A_2, \ldots, A_m$ have non-overlapping edges, they could be viewed as disjoint datasets of edges. Thus, the combination of $M_\varepsilon(A_1)$, $M_\varepsilon(A_2), \ldots, M_\varepsilon(A_m)$ is ε-edge differentially private. □

3) Proof of Theorem 3:

Proof. Under the transductive setting, since the same perturbed matrix $\widetilde{A}_{V^{(T)}}$ is used in both training and inference, the inference step does not leak extra graph structure information other than that in the DP GCN model. Therefore, the inference step is ε-edge differentially private due to the post-processing property of differential privacy. Under the inductive setting, the perturbation method M_ε is applied to both $A_{V^{(T)}}$ and $A_{V^{(I)}}$. Since $A_{V^{(T)}}$ and $A_{V^{(I)}}$ contain non-overlapping sets of edges, Lemma 1 guarantees ε-edge differential privacy of the `Inference` step. □

4) Proof of Theorem 4:

Proof. By definition, EDGERAND is ε-edge differentially private iff. for all symmetric matrices $A' \sim A \in \mathcal{A}$ and all subsets $S \subseteq \mathcal{A}$, Inequality (2) holds.

We first note that $\mathcal{M}$ operates on each cell A_{ij} independently. Therefore, the probability of perturbing the matrix A to get a certain output is the product of the probability of perturbing each cell in A to match the corresponding cell in the desired output. Let $e = (u, v)$ be the only differing edge between A and A'. For all $(i, j) \neq (u, v)$, $A_{ij} = A'_{ij}$, so the probability of perturbing A_{ij} and A'_{ij} into the same value is the same. For (u, v), however, getting the same outcome means that one of $A_{u,v}$ and A'_{uv} is changed through perturbation while the other remains unchanged. The probability of the state change, according to Algorithm 3, is $s/2$. Putting the statements together, we derive

$$\frac{\Pr[\mathcal{M}(A) \in S]}{\Pr[\mathcal{M}(A') \in S]} = \prod_{i,j} \frac{\Pr[\mathcal{M}(A_{ij}) \in S_{ij}]}{\Pr[\mathcal{M}(A'_{ij}) \in S_{ij}]} = \frac{\Pr[\mathcal{M}(A_{uv}) \in S_{uv}]}{\Pr[\mathcal{M}(A'_{uv}) \in S_{uv}]} \leq \frac{1 - s/2}{s/2} \leq \exp(\varepsilon)$$

when $\varepsilon \geq \ln\left(\frac{2}{s} - 1\right)$, $s \in (0, 1]$. □

5) Proof of Theorem 5:

Proof. In Algorithm 4, line 6 is ε_1-edge differentially private and line 7 is ε_2-edge differentially private due to the differential privacy guarantee of the Laplace mechanism. Therefore, from the composition theorem and the post-processing property of differential privacy, we know that LAPGRAPH guarantees ε-edge differential privacy. □

C. Proof of Theorem 6

The work on membership privacy [39] shows that differential privacy is a type of positive membership privacy, which prevents the attacker from significantly improving its confidence on a membership inference attack. Similarly, edge differential privacy bounds an attacker's precision in an edge re-identification attack. In this section, we present a formal proof for the upper bound defined in Theorem 6.

Proof. With a slight abuse of notations, we use e_1 to represent $A_{uv} = 1$, e_0 to represent $A_{uv} = 0$, and $\mathcal{R}_G$ to represent the attack $\mathcal{R}_{G_{BB}}(u, v)$ where G_{BB} is a black-box GCN. Based on Bayes' theorem, we have

$$\begin{aligned} &\Pr[e_1 \mid \mathcal{R}_G = 1] \\ =&\frac{\Pr[\mathcal{R}_G = 1 \mid e_1] \Pr[e_1]}{\Pr[\mathcal{R}_G = 1 \mid e_1] \Pr[e_1] + \Pr[\mathcal{R}_G = 1 \mid e_0] \Pr[e_0]} \\ =&\frac{\Pr[\mathcal{R}_G = 1 \mid e_1] \Pr[e_1]}{\Pr[\mathcal{R}_G = 1 \mid e_1] \Pr[e_1] + \Pr[\mathcal{R}_G = 1 \mid e_0] (1 - \Pr[e_1])}. \end{aligned} \tag{5}$$

Without loss of generality, let $\mathcal{G}$ represent the set of all GCN models. Since the attacker tries to re-identify the edges through querying the black-box model G, we could rewrite $\Pr[\mathcal{R}_G = 1 \mid e_1]$ as follows:

$$\Pr[\mathcal{R}_G = 1 \mid e_1] = \sum_{G_i \in \mathcal{G}} \Pr[\mathcal{R}_G = 1 \mid G = G_i] \Pr[G = G_i \mid e_1].$$

Similarly,

$$\Pr[\mathcal{R}_G = 1 \mid e_0] = \sum_{G_i \in \mathcal{G}} \Pr[\mathcal{R}_G = 1 \mid G = G_i] \Pr[G = G_i \mid e_0].$$

Therefore, to calculate the upper bound for Eq. 5, it is sufficient to upper bound the following ratio for any $G_i \in \mathcal{G}$:

$$\frac{\Pr[G = G_i \mid e_1] \Pr[e_1]}{\Pr[G = G_i \mid e_1] \Pr[e_1] + \Pr[G = G_i \mid e_0] (1 - \Pr[e_1])}. \tag{6}$$

Suppose $\mathcal{A}$ is the set of all possible adjacency matrices. Let $A^{(1)}, A^{(0)} \in \mathcal{A}$ be a pair of neighboring adjacency matrices differing by edge (u, v), and $A^{(1)}_{uv} = 1$, $A^{(0)}_{uv} = 0$. Based on the definition of differential privacy (Definition 3), for any $G_i \in \mathcal{G}$ and $A^{(1)}, A^{(0)} \in \mathcal{A}$, we have

$$\Pr[G = G_i \mid A = A^{(1)}] \leq \exp(\varepsilon) \Pr[G = G_i \mid A = A^{(0)}].$$

Therefore, for any $G_i \in \mathcal{G}$,

$$\Pr[G = G_i \mid e_1] \leq \exp(\varepsilon) \Pr[G = G_i \mid e_0].$$

Algorithm 2: Training and inference of DP GCN

Input: perturbation method $M \in \{$EDGERAND,LAPGRAPH $\}$, privacy parameter ε; node set $V^{(T)}$, $V^{(I)}$, adjacency matrix $A_{V^{(T)}}$, $A_{V^{(I)}}$, feature matrix $X^{(T)}$, $X^{(I)}$, and labels $y^{(T)}$. The subscript $^{(T)}$ stands for training and $^{(I)}$ is for inference.

Procedure `Perturbation(`$A_{V^{(T)}}$, M, ε`)`**:**
  $\widetilde{A}_{V^{(T)}} \leftarrow M_\varepsilon(A_{V^{(T)}})$
Procedure `Training(`$\widetilde{A}$, $V^{(T)}$, $X^{(T)}$, $y^{(T)}$`)`**:**
  GCN $\leftarrow$ a trained model using $\widetilde{A}_{V^{(T)}}, X^{(T)}, y^{(T)}$

if $V^{(T)} = V^{(I)}$ **then**
  $\widetilde{A}_{V^{(I)}} \leftarrow \widetilde{A}_{V^{(T)}}$
else
  $\widetilde{A}_{V^{(I)}} \leftarrow M_\varepsilon(A_{V^{(I)}})$
Function `Inference(`$\widetilde{A}, V^{(I)}, X^{(I)}$`)`**:**
  return GCN$(\widetilde{A}_{V^{(I)}}, X^{(I)}, \{W^i\})$

Algorithm 3: Edge Randomization (EDGERAND)

Input: a symmetric matrix A, privacy parameter s, randomization generator
Output: the perturbed outcome $\widetilde{A}$

Reset $\widetilde{A}$ to an all-zero matrix
for $1 \le i < j \le n$ **do**
  $x \leftarrow$ a sample drawn from Bern$(1-s)$
  if $x = 1$ **then**
    $\widetilde{A}_{ij}$ and $\widetilde{A}_{ji}$ are set to A_{ij} ▷ `Preservation`
  else
    $y \leftarrow$ a sample drawn from Bern$(1/2)$
    $\widetilde{A}_{ij}$ and $\widetilde{A}_{ji}$ are set to y ▷ `Randomization`

Since $\exp(\varepsilon) > 1$ for any positive privacy budget ε, we also have

$$\Pr[G = G_i \mid e_1] \le \exp(\varepsilon) \Pr[G = G_i \mid e_1].$$

Therefore,

$$\begin{aligned}
&\Pr[G = G_i \mid e_1] \\
\le &\exp(\varepsilon) \cdot \min(\Pr[G = G_i \mid e_0], \Pr[G = G_i \mid e_1]) \\
\le &\exp(\varepsilon) \left(\Pr\left[G = G_i \mid e_1\right] \Pr\left[e_1\right] + \Pr\left[G = G_i \mid e_0\right] \left(1 - \Pr\left[e_1\right]\right)\right)
\end{aligned}$$

The second inequality holds because $\le 0 \Pr[e] \le 1$. Therefore, we could compute the upper bound for the ratio in (6):

$$\begin{aligned}
&\frac{\Pr\left[G = G_i \mid e_1\right] \Pr\left[e_1\right]}{\Pr\left[G = G_i \mid e_1\right] \Pr\left[e_1\right] + \Pr\left[G = G_i \mid e_0\right] \left(1 - \Pr\left[e_1\right]\right)} \\
&\qquad \le \exp(\varepsilon) \cdot \Pr\left[e_1\right]
\end{aligned}$$

Because the graph density over $V^{(C)}$ is $k^{(C)}$, by the definition of graph density, we have $\Pr\left[e_1\right] = k^{(C)}$. Therefore,

$$\Pr\left[e_1 \mid \mathcal{R}_G = 1\right] \le \exp(\varepsilon) \cdot k^{(C)}$$

□

D. Detailed Algorithms for DP GCN

1) Algorithm for the Training and Inference of DP GCN: Algorithm 2 presents the perturbation, training, and inference steps in a differentially private GCN framework. $V^{(T)}$ and $A_{V^{(T)}}$ represent the set of training nodes and the adjacency matrix of the training graph; $V^{(I)}$ and $A_{V^{(I)}}$ represent the set of testing nodes and the adjacency matrix of the testing graph.

Algorithm 4: Laplace Mechanism for Graphs (LAPGRAPH)

Input: a symmetric matrix A, privacy parameter ε, randomization generator
Output: the perturbed outcome $\widetilde{A}$

$\varepsilon_1 \leftarrow 0.01\varepsilon$ ▷ `Distribute privacy budget`
$\varepsilon_2 \leftarrow \varepsilon - \varepsilon_1$
$T \leftarrow$ number of edges in A
$T \leftarrow T + \text{Lap}(1/\varepsilon_1)$ ▷ `Get a private count`
$A \leftarrow$ the upper triangular part of A
for $1 \le i < j \le n$ **do**
  $A_{ij} \leftarrow A_{ij} + \text{Lap}(1/\varepsilon_2)$ ▷ `Laplace mechanism`
▷ `Postprocess: Keep only the largest` T `cells`
$S \leftarrow$ the indice set for the largest T cells in A
Reset $\widetilde{A}$ to an all-zero matrix
for $(i, j) \in S$ **do**
  $\widetilde{A}_{ij}$ and $\widetilde{A}_{ji}$ are set to 1

The DP guarantee holds for both transductive training (*i.e.*, $V^{(T)} = V^{(I)}$) and inductive training (*i.e.*, $V^{(T)} \neq V^{(I)}$).

2) Algorithm for the DP Mechanisms: Algorithm 3 presents the algorithm for EDGERAND. We first randomly choose the cells to perturb and then randomly choose the target value from $\{0, 1\}$ for each cell to be perturbed.

Algorithm 4 presents the algorithm for LAPGRAPH. A small portion of the privacy budget ε_1 is used to compute the number of edges in the graph using the Laplacian mechanism, and the remaining privacy budget $\varepsilon_2 = \varepsilon - \varepsilon_1$ is used to apply Laplacian mechanism on the entire adjacency matrix. To preserve the degree of the original graph, the top-elements in the perturbed adjacency matrix are set to 1 and the remaining elements are set to 0.

E. Additional Discussions on the LINKTELLER Attack

1) Stealthiness and Alternative Detection Strategies: Our LINKTELLER attack queries the same set of inference nodes $V^{(I)}$ for $2n$ times where $n = \left|V^{(C)}\right|$, with the node features of one node slightly altered in each query. This abnormal behavior can easily distinguish LINKTELLER from a benign user and therefore allows the detection of the attack.

In particular, we describe details of potential detection strategies as follows. First, a defender can use validation data to evaluate both the attack and benign query performance in terms of the attack F1 score and query node classification accuracy under different query limits. Then the defender could optimize a query limit Q which decreases the attack performance while maintaining reasonable benign query accuracy. Such a query limit would depend on the properties of different datasets and how safety-critical the application is. Note that in general limiting the number of queries will not affect the performance for a single user, while it would hurt if several users aim to query about the same set of nodes, thus the query limit could be made for each node. In practice, the defender can directly flag the users who try to exceed the query limit Q for a limited set of nodes as suspicious for further inspection.

2) Estimation of the Density Belief $\hat{k}$: In this part, we describe a few actionable strategies for the attacker given limited knowledge of the density k and/or strategies to improve the accuracy of the density belief. For example, the attacker could

use some similar publicly available graphs (*e.g.*, a similar social network) or partial graphs to estimate k. Specifically:

(a) The attacker could estimate k based on partial graph information. With the prior knowledge of some connected/unconnected pairs, the attacker can calculate the influence values for each known pair. Then, she can estimate a threshold for distinguishing the connected pairs from the unconnected ones with high confidence, and thus obtain the estimated density belief $\hat{k}$.

(b) The attacker could estimate k based on the relationship of one or a few particular nodes. The attacker can start from an intentionally low $\hat{k}$ and increase it until an edge is inferred for the relationship, or until the known existing edges are inferred. The attacker then stops at this specific $\hat{k}$ and takes it as the density belief.

(c) The attacker could estimate k by running a link prediction algorithm. When a partial graph is available, the attacker can run a link prediction algorithm, *e.g.*, training a link prediction model, to predict all edges in the graph. Based on the predictions, the attacker will then obtain a rough estimate of the density belief $\hat{k}$ for use in LINKTELLER.

3) Variations of Our Attack under Different Settings: We discuss the variations of our attack under different settings, more specifically, different attacker's capabilities or different assumptions on the interaction model. We present three specific settings below.

When the attacker has additional knowledge of some edges: The attacker's prior knowledge on the existence of some edges can be leveraged to improve the density belief $\hat{k}$ in our LINKTELLER. More concretely, based on the knowledge of some edges, the attacker can calculate their influence values. Then, she can estimate a threshold for distinguishing the edges from the unconnected pairs with high confidence, and then obtain a refined estimation of the density belief $\hat{k}$. The attack effectiveness will subsequently be improved.

When the attacker has only partial control over a subset of node features: In this setting, part of the feature information is lost, and thus the accuracy of the estimation of the influence value would be negatively impacted, leading to the decrease of attack performance. However, how much the attack effectiveness will degrade also depends on the importance of the missing features.

When logits are not available: It is not straightforward to adapt our LINKTELLER to handle the case where logits are not available, which belongs to the "decision based blackbox attack category" rather than the score based. There are a few works [49], [50] in the image domain that perform certain decision-based blackbox attacks. However, how to estimate the gradient/influence value in GNNs based on decision only remains an interesting future direction.

4) Limitations to Overcome in Adapting LINKTELLER*:* First and foremost, in order to achieve high attack effectiveness, we need to derive exact influence calculations for different GNN structures specifically. We believe that our influence analysis based framework has the potential to perform well on different GNN structures with the influence value calculation tailored to each of them. Another potential obstacle in the adaptation is that LINKTELLER cannot deal with randomized models, such as the aggregation over sampled neighbors in GraphSAGE [14]. It could be an interesting future work to take such randomness into account for influence calculation.

5) Analysis on the Performance of LINKTELLER *Compared with Baselines:* First of all, we note that LSA2-X [11] relies on measuring certain distances based on either posteriors (of the node classification model) or node attributes to predict the connections. However, node classification and edge inference (*i.e.*, privacy attack goal here) are two distinct tasks, and node features (or posteriors) are useful for node classification does not mean that they will be useful for edge inference. Thus, LSA2-X which tries to provide the attacker with different levels of node information as prior knowledge to perform the edge re-identification attack is not effective. On the contrary, LINKTELLER tries to analyze the influence between nodes, which reflects the edge connection information based on our theoretical analysis (Theorem 1) and is indeed more effective for edge inference as we show empirically in Table I and Table II.

We point out that, to our best knowledge, there are no such settings where LINKTELLER may fail but other existing approaches (*e.g.*, LSA2-X) may succeed. The detailed reasons are provided above. To summarize, our LINKTELLER leverages the edge influence information, which is more relevant for the task of edge re-identification attack than purely node level information used in LSA2-X. We then discuss two specific scenarios below.

If the model makes inferences on single nodes and not subgraphs: In this case, LINKTELLER cannot obtain influence information between nodes of interests, and thus the edge re-identification performance would be less effective. Similarly for the baselines, where they would fail to calculate the statistics of a set of nodes to compare their similarity. That is to say, if the model makes inferences on single nodes, both LINKTELLER and baselines may fail to effectively attack, while LINKTELLER may still outperform baselines given that it leverages the influence value of edges explicitly.

If the inference is transductive vs. inductive: We first point out that the inductive setting is more challenging than the transductive setting. We then analyze the potential performance of LINKTELLER in the transductive setting. LINKTELLER is naturally applicable to the transductive setting– the attacker may happen to query the node in the training graph. Since these nodes are involved in model training, the influence value and the rank may be more accurate, leading to even better attack performance. As shown in the experimental results in Appendix G5, LINKTELLER indeed outperforms the baselines as well in the transductive setting.

F. Details of Evaluation

1) Dataset Statistics: We provide the dataset statistics in Table IV. The three datasets (Cora, Citeseer, and Pubmed) in the transductive setting are all citation networks. Concretely, the nodes are documents/publications and the edges are the

TABLE IV: Dataset statistics ("m" represents **m**ulti-label classification; "s" represents **s**ingle-label.)

(a) Datasets in the **inductive** setting

Dataset	Nodes	Edges	Classes	Features
Twitch-ES	4,648	59,382	2 (s)	3,170
Twitch-RU	4,385	37,304	2 (s)	3,170
Twitch-DE	9,498	153,138	2 (s)	3,170
Twitch-FR	6,549	112,666	2 (s)	3,170
Twitch-ENGB	7,126	35,324	2 (s)	3,170
Twitch-PTBR	1,912	31,299	2 (s)	3,170
PPI	14,755	225,270	121 (m)	50
Flickr	89,250	899,756	7 (s)	500

(b) Datasets in the **transductive** setting

Dataset	Nodes	Edges	Classes	Features
Cora	2,708	5,429	7 (s)	1,433
Citeseer	3,327	4,732	6 (s)	3,703
Pubmed	19,717	44,338	3 (s)	500

citation links between them. The node features are the sparse bag-of-words feature vectors for each document.

2) Evaluation Metrics for Model Utility: We describe how we evaluate the utility of the trained models, including the vanilla GCN models, two DP GCN models (EDGERAND and LAPGRAPH), and the MLP models.

We apply slightly different evaluation metrics across datasets given their varying properties. The twitch datasets are for binary classification tasks on imbalanced datasets. Therefore, we use **F1 score of the rare class** to measure the utility of the trained GCN model. To compute the value, we first identify the minority class in the dataset and then view it as the positive class for the calculation of the F1 score. During training, we train on twitch-ES; during inference, we evaluate the trained model on twitch-{RU, DE, FR, ENGB, PTBR}. For PPI and Flickr datasets where there is no significant class imbalance, we follow previous works [14], [16] and use **micro-averaged F1 score** to evaluate the classification results.

For DP GCNs particularly, in each setting, we report the averaged results over 10 runs that use different random seeds for noise generation.

3) Normalization Techniques: We followed Rong *et al.* [15] and experimented with the techniques provided below. A is an adjacency matrix $\in \{0,1\}^{n\times n}$, $D = A + I$, and $\widehat{A}$ is the normalized matrix.

$$\widehat{A} = I + D^{-1/2}AD^{-1/2} \tag{7}$$

$$\widehat{A} = (D+I)^{-1/2}(A+I)(D+I)^{-1/2} \tag{8}$$

$$\widehat{A} = I + (D+I)^{-1/2}(A+I)(D+I)^{-1/2} \tag{9}$$

$$\widehat{A} = (D+I)^{-1}(A+I) \tag{10}$$

- `FirstOrderGCN`: First-order GCN (Eq. 7)
- `AugNormAdj`: Augmented Normalized Adjacency (Eq. 8)
- `BingGeNormAdj`: Augmented Normalized Adjacency with Self-loop (Eq. 9)
- `AugRWalk`: Augmented Random Walk (Eq. 10)

4) Search Space for the Hyper-parameters: In training the models, we perform an extensive grid search to find the best set of hyper-parameters. We describe the search space of the hyper-parameters below.

- learning rate (lr): {0.005, 0.01, 0.02, 0.04, 0.05, 0.1, 0.2}
- dropout rate: {0.05, 0.1,0.2,0.3,0.5,0.8}
- number of GCN layers: {1,2,3}
- number of hidden units: {64,128,256,512}
- normalization technique: {FirstOrderGCN, AugNormAdj, BingGeNormAdj, AugRWalk}

5) Best Hyper-parameters for the Vanilla-GCN: Below, we describe the best combinations we achieve for Vanilla-GCN models. For twitch-ES, we use the method First-Order GCN to normalize the input graph. We train a two-layer GCN with the number of hidden units 256. The dropout rate is set to 0.5 and the learning rate is 0.01. The training epoch is 200 and the model converges within 200 epochs. For PPI, we use Augmented Normalized Adjacency with Self-loop for normalizing the adjacency matrix and train a two-layer GCN with the number of hidden layers 256. The dropout rate is 0.4 and the learning rate is 0.05. The training epoch is 200 where the model converges. For Flickr, we use Augmented Normalized Adjacency for normalization and train a two-layer GCN with the number of hidden layers 256. The dropout rate is 0.2 and the learning rate is 0.0005. The number of epochs is 200 within which the model converges.

6) Best Hyper-parameters for the Vanilla-GAT: We use 3-layer GATs for both PPI and Flickr datasets as described in Section V-F. For PPI, the number of heads per layer are 6, 6, and 6 for the three layers. The hidden layer dimensions are 256 and 256. The skip connection is added. During training, we use dropout rate of 0; test accuracy on the unseen node set is 0.66. For Flickr, the number of heads per layer are 4, 4, and 4 for the three layers. The hidden layer dimensions are 256 and 256. The skip connection is added. During training, we use dropout rate of 0.5; test accuracy on the unseen node set is 0.47.

TABLE V: **Precision** (%) of the *random attack* baseline.

Degree	Dataset						
	RU	DE	FR	ENGB	PTBR	PPI	Flickr
low	1.7e-2	6.7e-3	7.5e-3	1.3e-2	4.5e-2	1.8e-2	4.0e-3
unconstrained	4.3e-1	3.2e-1	5.3e-1	1.5e-1	1.6	2.0e-1	1.0e-2
high	1.4	7.5e-1	1.0	9.5e-1	3.4	1.2	2.6e-1

G. More Evaluation Results

1) Results for the Random Attack Baseline: As described in Section V-C2, for a random classifier with Bernoulli parameter p, given a set of instances containing a positive examples and b negative examples, its precision is $a/(a+b)$ and recall is p, which are *density* k and *belief density* $\hat{k}$, respectively. We present the precision scores of the random classifier in Table V. Compared with Table I, wee see that the *precision* of LINKTELLER is much higher than the random attack baseline. This reveals the significant advantage an attacker is able to gain through querying an inference API, which may lead to severe privacy loss. As for the *recall* which is equal to density belief, the number $\hat{k} \in \{k/4, k/2, k, 2k, 4k\}$ is also extremely

TABLE VI: **Attack Performance (Precision and Recall)** of LINKTELLER on twitch datasets, evaluated against a *3-layer GCN*. Each table corresponds to a dataset. We sample nodes of low, unconstrained, and high degrees as our targets. Groups of rows represent different *density belief* $\hat{k}$ of the attacker.

RU	low		unconstrained		high	
$\hat{k}$	precision	recall	precision	recall	precision	recall
$k/4$	100.0 ± 0.0	33.0 ± 2.8	80.8 ± 4.2	22.1 ± 1.5	83.9 ± 2.1	15.4 ± 1.5
$k/2$	84.6 ± 0.0	51.9 ± 4.3	65.1 ± 2.1	35.5 ± 0.9	72.9 ± 1.1	26.7 ± 1.9
k	69.3 ± 8.2	81.1 ± 4.2	45.7 ± 2.2	50.0 ± 2.8	55.6 ± 2.8	40.7 ± 1.6
$2k$	40.7 ± 5.0	95.0 ± 4.3	27.7 ± 1.8	60.4 ± 2.7	37.4 ± 2.9	54.6 ± 1.0
$4k$	20.3 ± 2.5	95.0 ± 4.3	15.8 ± 1.0	68.8 ± 3.0	23.0 ± 2.4	67.0 ± 2.6

DE	low		unconstrained		high	
$\hat{k}$	precision	recall	precision	recall	precision	recall
$k/4$	91.7 ± 11.8	29.0 ± 3.4	75.2 ± 5.8	18.1 ± 2.6	71.3 ± 6.7	18.1 ± 1.8
$k/2$	82.1 ± 12.7	49.6 ± 6.3	54.6 ± 2.6	26.3 ± 3.0	50.3 ± 4.6	25.5 ± 2.7
k	64.6 ± 7.6	73.0 ± 9.0	32.7 ± 1.0	31.3 ± 3.0	33.0 ± 2.4	33.4 ± 2.4
$2k$	41.7 ± 3.6	88.9 ± 7.9	20.4 ± 0.2	38.9 ± 2.6	21.9 ± 1.5	44.5 ± 3.1
$4k$	22.4 ± 1.7	94.4 ± 7.9	13.9 ± 0.4	53.1 ± 2.1	14.0 ± 0.6	56.7 ± 1.4

FR	low		unconstrained		high	
$\hat{k}$	precision	recall	precision	recall	precision	recall
$k/4$	100.0 ± 0.0	28.3 ± 2.4	85.4 ± 5.6	19.9 ± 1.6	87.9 ± 3.8	21.3 ± 1.6
$k/2$	100.0 ± 0.0	50.0 ± 0.0	71.5 ± 5.5	33.3 ± 2.9	70.9 ± 8.5	34.0 ± 1.5
k	78.3 ± 2.4	78.3 ± 2.4	50.1 ± 5.1	46.6 ± 5.0	48.6 ± 10.1	46.2 ± 5.3
$2k$	41.7 ± 2.4	81.7 ± 6.2	29.1 ± 2.1	54.1 ± 4.0	28.6 ± 6.0	54.4 ± 6.4
$4k$	20.8 ± 1.2	81.7 ± 6.2	16.3 ± 1.2	60.7 ± 5.4	16.6 ± 2.9	63.4 ± 5.4

ENGB	low		unconstrained		high	
$\hat{k}$	precision	recall	precision	recall	precision	recall
$k/4$	91.7 ± 11.8	27.7 ± 5.2	83.1 ± 3.3	22.9 ± 4.4	86.7 ± 1.1	22.1 ± 0.4
$k/2$	85.7 ± 11.7	47.0 ± 12.7	69.1 ± 5.0	37.4 ± 6.1	71.1 ± 3.0	36.2 ± 1.7
k	66.4 ± 9.3	68.4 ± 18.9	48.7 ± 6.9	51.7 ± 5.9	50.7 ± 3.0	51.6 ± 3.3
$2k$	46.0 ± 7.5	89.0 ± 4.4	29.6 ± 4.5	62.7 ± 6.9	30.7 ± 1.7	62.4 ± 3.8
$4k$	23.7 ± 3.7	91.6 ± 3.1	17.0 ± 3.6	70.9 ± 3.4	17.5 ± 0.4	71.2 ± 1.9

PTBR	low		unconstrained		high	
$\hat{k}$	precision	recall	precision	recall	precision	recall
$k/4$	100.0 ± 0.0	26.7 ± 1.3	80.8 ± 4.7	20.9 ± 4.3	86.7 ± 1.9	21.3 ± 2.1
$k/2$	91.8 ± 5.8	48.6 ± 5.7	65.5 ± 7.7	33.3 ± 5.3	73.5 ± 2.0	36.2 ± 3.4
k	71.1 ± 0.9	74.3 ± 4.0	46.3 ± 9.5	46.0 ± 4.1	53.5 ± 2.3	52.5 ± 3.9
$2k$	41.3 ± 1.3	85.6 ± 2.8	30.2 ± 7.4	59.5 ± 3.4	32.9 ± 2.3	64.4 ± 4.1
$4k$	21.4 ± 1.1	88.7 ± 0.6	18.2 ± 4.8	71.4 ± 3.5	19.1 ± 1.5	74.7 ± 3.2

TABLE VII: **AUC** of LINKTELLER on twitch datasets, evaluated against a *3-layer GCN*. Each column corresponds to one dataset. Rows represent sampled nodes of varying degrees.

Degree	Dataset				
	RU	DE	FR	ENGB	PTBR
low	1.00 ± 0.00	0.99 ± 0.02	0.94 ± 0.05	1.00 ± 0.00	0.98 ± 0.00
uncon-strained	0.96 ± 0.01	0.92 ± 0.01	0.91 ± 0.02	0.97 ± 0.01	0.92 ± 0.01
high	0.93 ± 0.01	0.88 ± 0.01	0.90 ± 0.01	0.95 ± 0.00	0.90 ± 0.00

small compared with LINKTELLER. To sum up, LINKTELLER significantly outperforms the random baseline.

2) Results for a 3-layer GCN: In Section V-E in the main paper, we mainly evaluated 2-layer GCNs. In this section, we evaluate the performance of LINKTELLER on 3-layer GCNs to provide a more comprehensive view of LINKTELLER's capability.

a) Model: For training the models, we follow the same principle described in Section V-B and use the same search space as in Appendix F4. The best combination of hyper-parameters/configurations are described below. We use the method First-Order GCN to normalize the input graph. The hidden layer dimensions are 64 and 64. The dropout rate is set to 0.5 and the learning rate is 0.01. The training epoch is 50 and the model converges. The test F1 score on twitch-{RU, DE, FR, ENGB, PTBR} are 0.3419, 0.4698, 0.4926, 0.6027, and 0.5198, respectively.

b) Attack Results: We present the attack results of LINKTELLER on the 3-layer GCN in Table VI and Table VII. Comparing Table VI with Table I, and Table VII with Table II, we see that the performance of LINKTELLER on the 3-layer GCN only drops a little. For 1-layer GCN, we know from Proposition 1 that LINKTELLER can perform a perfect attack. For GCNs with more than 3 layers, we did not bother to evaluate the attack performance since deeper GCNs suffer from over-smoothing [51] and give poor classification results. Thus, we can confidently conclude that LINKTELLER is a successful attack against most practical GCN models.

3) Running Time of LINKTELLER*:* We report the running time of LINKTELLER on vanilla GCNs in Table VIII, corresponding to the experiments in Section V in the main paper. As the table shows, LINKTELLER is a highly efficient attack. On DP GCNs using EDGERAND mechanism, when the graph becomes denser under smaller privacy budgets, one forward pass of the network takes longer, since the cost of matrix computation becomes larger. However, the increase of running time reflected in the attack time is only marginal, so we omit the running time for DP GCNs here. Overall, LINKTELLER can efficiently and effectively attack both vanilla GCNs and DP GCNs.

4) More Results for LINKTELLER *on vanilla GCNs and DP GCNs:* First of all, we present the additional evaluation results for LINKTELLER on vanilla GCNs corresponding to Section V. The results are presented in Table IX, which are of the same format as Table I.

Next, we show the comprehensive evaluation results on a combination of 2 DP mechanisms (EDGERAND and LAPGRAPH), 10 privacy budgets $(1.0, 2.0, \ldots, 10.0)$, 3 sampling strategies (low degree, unconstrained degree, high degree), and 5 density beliefs $(k/4, k/2, k, 2k, 4k)$. We present the results in Tables XIII to XXVI. The 3 subtables in each table correspond to the 3 sampling strategies.

In Section VI of the main paper, we present the results for density belief $\hat{k} = k$. Here, we look at the results for other inexact $\hat{k}$ values and find that similar observations hold. First, the effectiveness of LINKTELLER will decrease as a result of increasing privacy guarantee; while when the guarantee is not

TABLE VIII: **Running time** of LINKTELLER on vanilla GCNs corresponding to experiments in Section V. The time unit is "second".

Degree	Dataset						
	RU	DE	FR	ENGB	PTBR	PPI	Flickr
low	1.25 ± 0.0	1.61 ± 0.1	1.32 ± 0.0	1.28 ± 0.0	1.12 ± 0.1	1.48 ± 0.1	3.08 ± 0.1
uncon-strained	1.24 ± 0.1	1.60 ± 0.1	1.34 ± 0.1	1.29 ± 0.1	1.10 ± 0.1	1.47 ± 0.1	3.07 ± 0.2
high	1.24 ± 0.1	1.61 ± 0.0	1.30 ± 0.2	1.26 ± 0.0	1.15 ± 0.4	1.48 ± 0.0	3.05 ± 0.2

TABLE IX: **Attack Performance** of LINKTELLER on additional datasets, compared with two baseline methods LSA2-{post, attr} [11]. Each table corresponds to a dataset. We sample nodes of low, unconstrained, and high degrees as our targets. Groups of rows represent different *density belief* $\hat{k}$ of the attacker.

twitch-DE		low		unconstrained		high	
$\hat{k}$	Method	precision	recall	precision	recall	precision	recall
$k/4$	Ours	**83.3** ± 23.6	**26.2** ± 7.0	**94.0** ± 1.3	**22.5** ± 1.3	**99.0** ± 0.4	**25.2** ± 1.2
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.1 ± 0.2	0.0 ± 0.1
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	0.4 ± 0.5	0.1 ± 0.1	2.6 ± 1.1	0.7 ± 0.3
$k/2$	Ours	**91.7** ± 11.8	**55.2** ± 3.7	**92.4** ± 4.1	**44.2** ± 1.6	**96.9** ± 0.3	**49.2** ± 2.5
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.1 ± 0.1	0.0 ± 0.1
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	0.9 ± 0.3	0.4 ± 0.1	2.5 ± 0.8	1.3 ± 0.5
k	Ours	**81.8** ± 4.8	**92.5** ± 5.9	**81.2** ± 6.6	**77.2** ± 3.4	**79.2** ± 1.1	**80.4** ± 3.1
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.1
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	1.2 ± 0.3	1.2 ± 0.4	1.8 ± 0.4	1.8 ± 0.5
$2k$	Ours	**44.7** ± 3.4	**95.2** ± 6.7	**49.0** ± 4.7	**93.1** ± 3.2	**46.9** ± 2.0	**95.0** ± 0.9
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.1	0.1 ± 0.1	0.0 ± 0.0	0.0 ± 0.1
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	0.8 ± 0.1	1.6 ± 0.2	1.4 ± 0.3	2.8 ± 0.8
$4k$	Ours	**23.8** ± 0.3	**100.0** ± 0.0	**25.8** ± 1.9	**98.1** ± 0.7	**24.3** ± 1.1	**98.5** ± 0.5
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.1	0.2 ± 0.2	0.0 ± 0.0	0.0 ± 0.1
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	0.7 ± 0.1	2.7 ± 0.4	1.1 ± 0.2	4.4 ± 0.9

twitch-ENGB		low		unconstrained		high	
$\hat{k}$	Method	precision	recall	precision	recall	precision	recall
$k/4$	Ours	**100.0** ± 0.0	**30.3** ± 4.1	**92.6** ± 3.0	**25.5** ± 4.9	**98.8** ± 0.5	**25.2** ± 0.2
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.1 ± 0.2	0.0 ± 0.0
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1.5 ± 0.4	0.4 ± 0.1
$k/2$	Ours	**100.0** ± 0.0	**54.2** ± 8.7	**84.3** ± 5.6	**45.6** ± 7.6	**96.0** ± 1.2	**48.9** ± 0.6
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.1 ± 0.1	0.0 ± 0.0
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	0.3 ± 0.4	0.2 ± 0.2	1.8 ± 0.3	0.9 ± 0.1
k	Ours	**83.1** ± 6.6	**84.0** ± 11.4	**67.9** ± 6.3	**72.9** ± 10.9	**81.6** ± 2.7	**83.1** ± 2.7
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.1 ± 0.0	0.1 ± 0.0
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	0.7 ± 0.2	0.7 ± 0.2	2.0 ± 0.1	2.0 ± 0.1
$2k$	Ours	**50.7** ± 8.2	**97.9** ± 2.9	**43.6** ± 9.0	**91.2** ± 4.9	**47.3** ± 0.7	**96.3** ± 1.3
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.4 ± 0.5	0.8 ± 1.0
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	0.5 ± 0.2	1.2 ± 0.6	1.7 ± 0.2	3.4 ± 0.3
$4k$	Ours	**26.0** ± 4.9	**100.0** ± 0.0	**23.5** ± 5.9	**97.3** ± 1.1	**24.2** ± 0.2	**98.5** ± 0.5
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.2 ± 0.2	0.8 ± 1.0
	LSA2-attr	0.0 ± 0.0	0.0 ± 0.0	0.4 ± 0.2	1.7 ± 1.0	1.7 ± 0.1	6.9 ± 0.3

twitch-PTBR		low		unconstrained		high	
$\hat{k}$	Method	precision	recall	precision	recall	precision	recall
$k/4$	Ours	**100.0** ± 0.0	**26.7** ± 1.3	**95.6** ± 1.6	**25.1** ± 6.5	**98.4** ± 1.3	**24.2** ± 2.5
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.1	0.0 ± 0.0
	LSA2-attr	4.6 ± 3.3	1.2 ± 0.9	4.7 ± 0.9	1.2 ± 0.4	6.9 ± 0.7	1.7 ± 0.1
$k/2$	Ours	**99.0** ± 1.5	**52.3** ± 3.3	**93.6** ± 1.4	**49.0** ± 12.0	**97.3** ± 1.6	**47.9** ± 4.7
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	LSA2-attr	2.4 ± 1.7	1.2 ± 0.9	4.4 ± 0.8	2.3 ± 0.8	6.3 ± 1.0	3.0 ± 0.3
k	Ours	**85.4** ± 2.2	**89.2** ± 4.2	**78.7** ± 8.7	**80.3** ± 13.1	**86.0** ± 5.8	**84.2** ± 4.0
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	LSA2-attr	1.2 ± 0.9	1.2 ± 0.9	4.3 ± 0.9	4.7 ± 1.7	6.0 ± 0.8	5.9 ± 0.5
$2k$	Ours	**48.3** ± 2.7	**100.0** ± 0.0	**48.7** ± 12.2	**95.7** ± 4.1	**50.2** ± 5.4	**97.9** ± 0.4
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	2.4 ± 1.7	4.7 ± 3.2
	LSA2-attr	0.6 ± 0.4	1.2 ± 0.9	3.8 ± 0.3	8.0 ± 2.4	5.0 ± 0.7	9.8 ± 0.6
$4k$	Ours	**24.1** ± 1.4	**100.0** ± 0.0	**25.5** ± 7.5	**99.0** ± 0.6	**25.5** ± 2.8	**99.5** ± 0.1
	LSA2-post	0.0 ± 0.0	0.0 ± 0.0	1.3 ± 0.9	6.5 ± 4.7	2.5 ± 0.2	10.1 ± 1.9
	LSA2-attr	0.6 ± 0.2	2.4 ± 0.7	3.1 ± 0.2	13.2 ± 3.6	4.1 ± 0.4	15.8 ± 0.5

sufficient (*i.e.*, ε is large), LINKTELLER is not weakened by much. Second, DP can provide better protection to nodes of low degrees. In addition, we note that our LINKTELLER is not sensitive to the density belief $\hat{k}$ and achieves non-negligible success rate for all $\hat{k}$.

5) LINKTELLER *in the Transductive Setting:* In the main paper, we mainly evaluate the performance of LINKTELLER in the *inductive* setting. As analyzed in Appendix E5, in the *transductive* setting, LINKTELLER is expected to achieve better performance and retain its advantage over LSA2-X. Here, we aim to provide evaluations on the performance of LINKTELLER in the *transductive* setting to support the analysis.

TABLE X: **Attack Performance (Precision and Recall)** of LINKTELLER on three datasets in the *transductive* setting, compared with two baseline methods LSA2-{post, attr} [11]. We follow He *et al.* [11] to compose a balanced dataset containing an equal number of connected and unconnected node pairs. Groups of rows represent different *density belief* $\hat{k}$ of the attacker.

		Cora		Citeseer		Pubmed	
$\hat{k}$	Method	precision	recall	precision	recall	precision	recall
$k/4$	Ours	**99.9** ± 0.1	**25.0** ± 0.0	**100.0** ± 0.0	**25.0** ± 0.0	**100.0** ± 0.0	**25.0** ± 0.0
	LSA2-post	96.7 ± 0.2	24.2 ± 0.0	98.8 ± 0.1	24.7 ± 0.0	89.9 ± 0.2	22.5 ± 0.1
	LSA2-feat	96.9 ± 0.2	24.2 ± 0.0	99.8 ± 0.1	24.9 ± 0.0	97.8 ± 0.2	24.4 ± 0.0
$k/2$	Ours	**99.9** ± 0.0	**50.0** ± 0.0	**100.0** ± 0.0	**50.0** ± 0.0	**100.0** ± 0.0	**50.0** ± 0.0
	LSA2-post	94.1 ± 0.3	47.0 ± 0.1	96.7 ± 0.0	48.4 ± 0.0	86.8 ± 0.1	43.4 ± 0.0
	LSA2-feat	90.4 ± 0.5	45.2 ± 0.2	97.4 ± 0.1	48.7 ± 0.1	95.2 ± 0.1	47.6 ± 0.0
k	Ours	**99.5** ± 0.1	**99.5** ± 0.1	**99.7** ± 0.0	**99.7** ± 0.0	**99.7** ± 0.0	**99.7** ± 0.0
	LSA2-post	86.7 ± 0.2	86.7 ± 0.2	90.1 ± 0.2	90.1 ± 0.2	78.8 ± 0.1	78.8 ± 0.1
	LSA2-feat	73.6 ± 0.1	73.6 ± 0.1	80.9 ± 0.1	80.9 ± 0.1	82.4 ± 0.1	82.4 ± 0.1
$1.5k$	Ours	**66.7** ± 0.0	**100.0** ± 0.0	**66.7** ± 0.0	**100.0** ± 0.0	**66.6** ± 0.0	**99.9** ± 0.0
	LSA2-post	66.0 ± 0.0	99.1 ± 0.0	66.4 ± 0.0	99.6 ± 0.0	65.3 ± 0.0	98.0 ± 0.0
	LSA2-feat	59.9 ± 0.2	89.8 ± 0.2	63.2 ± 0.1	94.7 ± 0.2	64.0 ± 0.0	96.0 ± 0.0

TABLE XI: **AUC** of LINKTELLER comparing with two baselines LSA2-{post, attr} [11] in the *transductive* setting. Each column corresponds to one dataset. Different rows represent different methods.

Method	Dataset		
	Cora	Citeseer	Pubmed
Ours	**1.00** ± 0.00	**1.00** ± 0.00	**1.00** ± 0.00
LSA2-post	0.93 ± 0.00	0.96 ± 0.00	0.87 ± 0.00
LSA2-attr	0.81 ± 0.00	0.89 ± 0.00	0.90 ± 0.00

We compare LINKTELLER to LSA2-{post, attr} [11] in the *transductive* setting using three datasets (Cora, Citeseer, and Pubmed) from their paper. We also follow the same setup (as in their Paragraph "Datasets Configuration" in Section 5.1) to compose the *balanced* set of node pairs to be attacked which contains an equal number of connected and unconnected pairs. We follow the hyper-parameters in Kipf *et al.* [13] to train the GCN models on these datasets and then perform LINKTELLER attack and LSA2-{post, attr} attacks on the trained models.

We report the attack performance (Precision and Recall) in Table X and the AUC scores in Table XI. **First**, as a sanity check, our results in Table XI matches the Figure 4 in He *et al.* [11] on these three datasets. **Second**, we evaluate the density belief $\hat{k}$ only up to $1.5k$ in Table X, since $2k$ corresponds to the case where all node pairs are predicted positive by the attacker, leading to 50% precision and 100% recall for *all* methods. **Overall**, as shown in the two tables, LINKTELLER invariably outperforms LSA2-{post, attr} in the transductive setting.

6) Choosing ε on a Validation Dataset: In Section VI, we present the model utility and attack effectiveness under a range of ε (1-10) in Figure 3, and provide corresponding discussions regarding the tradeoff between model utility and privacy as well as its dependencies in Section VI-C and Section VI-D.

In this part, we take on a new perspective and focus on a practical approach for selecting the appropriate parameters (mainly the privacy budget ε) to train DP GCN models with reasonable model utility–we select the parameter ε on the validation dataset, and then report the final performance on the testing set. Concretely, when selecting ε on the *validation set*,

TABLE XII: (a) **Model utility** and (b) **attack effectiveness** on different models. Each column corresponds to a dataset. We consider four types of models: vanilla GCN, MLP, EDGERAND, and LAPGRAPH.

(a) **Model utility** (F1 score)

Model	Dataset					
	RU	DE	FR	ENGB	PTBR	Flickr
vanilla GCN	0.319	0.551	0.404	0.601	0.411	0.515
MLP	0.290	0.545	0.373	0.598	0.358	0.463
EDGERAND	$\varepsilon = 7$					$\varepsilon = 6$
	0.299 ± 0.006	0.545 ± 0.003	0.321 ± 0.027	0.607 ± 0.000	0.423 ± 0.018	0.459 ± 0.002
LAPGRAPH	$\varepsilon = 8$					$\varepsilon = 9$
	0.292 ± 0.011	0.546 ± 0.001	0.299 ± 0.017	0.601 ± 0.001	0.401 ± 0.010	0.467 ± 0.002

(b) **Attack effectiveness** (F1 score) on different node degree distributions

Degree	Model	Dataset					
		RU	DE	FR	ENGB	PTBR	Flickr
low	vanilla GCN	84.9 ± 1.2	86.8 ± 5.1	92.5 ± 5.4	82.9 ± 4.9	86.6 ± 1.3	52.1 ± 5.8
	EDGERAND	18.9 ± 10.8	4.4 ± 6.3	0.0 ± 0.0	19.0 ± 12.0	32.9 ± 2.5	0.0 ± 0.0
	LAPGRAPH	22.9 ± 3.3	5.3 ± 7.5	13.3 ± 12.5	22.0 ± 1.0	23.8 ± 2.2	26.8 ± 10.1
unconstrained	vanilla GCN	74.7 ± 1.5	78.5 ± 4.5	80.9 ± 2.0	69.5 ± 2.5	77.9 ± 3.5	32.9 ± 13.3
	EDGERAND	58.1 ± 2.2	60.1 ± 5.0	67.1 ± 4.6	41.6 ± 8.1	73.2 ± 5.1	0.0 ± 0.0
	LAPGRAPH	59.6 ± 1.2	59.6 ± 2.0	67.1 ± 2.9	46.9 ± 3.7	68.4 ± 5.8	2.4 ± 3.4
high	vanilla GCN	75.8 ± 2.3	78.9 ± 0.8	83.0 ± 3.7	82.2 ± 3.4	84.8 ± 1.6	18.3 ± 5.2
	EDGERAND	72.6 ± 1.5	76.5 ± 1.8	78.1 ± 2.7	82.3 ± 1.5	83.7 ± 1.0	16.9 ± 2.9
	LAPGRAPH	69.6 ± 1.0	68.5 ± 1.1	73.4 ± 2.8	68.0 ± 1.7	78.4 ± 1.4	15.7 ± 2.6

we set a lower threshold (*e.g.*, F1 score of the MLP model) for the model utility, and select the smallest ε satisfying the condition that the utility of the trained DP GCN on the validation set is higher than the given threshold. Then, we evaluate both the model utility and attack effectiveness under the selected ε on the *testing set*.

We report the evaluation results in Table XII. We do not include results for the PPI dataset, since none of the DP GCN models could attain utility higher than the MLP model. (In fact, even the vanilla GCN model cannot match the performance of the MLP model, which have been concretely discussed in Section VI-D.) For the twitch datasets and Flickr dataset, we select $\varepsilon \in \{1.0, 2.0, \ldots, 10.0\}$ on the validation dataset and include the numbers in Table XII(a). Specifically, for the twitch datasets, the model is trained on twitch-ES (with the parameters selected on the validation set) and then directly transferred to twitch-{RU, DE, FR, ENGB, PTBR}, so we report only one number for all five countries.

We next discuss the conclusions from Table XII. **First**, regarding the model utility in Table XII(a), with the same constraint on model utility on the same dataset, we end up with different ε for different DP mechanisms, *e.g.*, $\varepsilon = 6$ for EDGERAND and $\varepsilon = 9$ for LAPGRAPH on the Flickr dataset. This implies that the level of noise that can be tolerated by different DP mechanisms is different. **Second**, in terms of the attack effectiveness in Table XII(b), the main conclusion is that the theoretical guarantee provided by DP cannot translate into sufficient protection against LINKTELLER while maintaining a reasonable level of model utility. Specifically, with $\varepsilon = 6$ for EDGERAND and $\varepsilon = 9$ for LAPGRAPH, the theoretical upper bound of precision is greater than 1 for all the datasets based on Theorem 6, which means that the DP GCN methods do *not* provide any reasonable *theoretical* privacy protection against LINKTELLER. **However**, the attack effectiveness presented in Table XII(b) suggests that, in low degree settings, differential privacy can *empirically* protect against LINKTELLER although the level of protection is heavily data-dependent, varying a lot across different *datasets* and different *node degree distributions*. Relevant discussions on the impact of node degree distributions specifically are provided in Section VI-D.

TABLE XIII: twitch-RU (EDGERAND)

low		$\hat{k}=k/4$	$\hat{k}=k/2$	$\hat{k}=k$	$\hat{k}=2k$	$\hat{k}=4k$
$\varepsilon=1$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon=2$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon=3$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon=4$	precision	0.0 ± 0.0	0.0 ± 0.0	1.3 ± 1.9	1.3 ± 1.9	1.3 ± 0.5
	recall	0.0 ± 0.0	0.0 ± 0.0	1.5 ± 2.1	3.0 ± 4.3	6.2 ± 2.1
$\varepsilon=5$	precision	4.8 ± 6.7	5.1 ± 7.3	4.0 ± 3.3	3.3 ± 1.9	2.7 ± 1.2
	recall	1.5 ± 2.1	3.0 ± 4.3	4.8 ± 3.7	7.7 ± 4.2	12.8 ± 6.0
$\varepsilon=6$	precision	19.0 ± 6.7	12.8 ± 3.6	9.3 ± 1.9	8.0 ± 2.8	6.7 ± 3.3
	recall	6.2 ± 2.1	7.7 ± 1.7	11.0 ± 2.0	18.7 ± 6.2	31.5 ± 15.1
$\varepsilon=7$	precision	28.6 ± 11.7	25.6 ± 13.1	17.3 ± 10.0	16.7 ± 4.7	16.7 ± 2.1
	recall	9.5 ± 3.9	16.0 ± 8.3	20.8 ± 11.9	39.9 ± 13.2	78.9 ± 12.8
$\varepsilon=8$	precision	38.1 ± 17.8	38.5 ± 10.9	38.7 ± 7.5	34.0 ± 1.6	20.7 ± 1.2
	recall	12.8 ± 6.0	24.0 ± 7.9	46.1 ± 11.5	80.3 ± 8.2	97.0 ± 2.1
$\varepsilon=9$	precision	71.4 ± 11.7	56.4 ± 3.6	57.3 ± 5.0	39.3 ± 1.9	21.3 ± 1.7
	recall	23.7 ± 4.4	34.5 ± 2.1	67.4 ± 6.1	92.7 ± 7.4	100.0± 0.0
$\varepsilon=10$	precision	95.2 ± 6.7	79.5 ± 3.6	73.3 ± 1.9	42.0 ± 3.3	21.3 ± 1.7
	recall	31.5 ± 4.0	48.6 ± 3.0	86.4 ± 5.9	98.5 ± 2.1	100.0± 0.0

unconstrained		$\hat{k}=k/4$	$\hat{k}=k/2$	$\hat{k}=k$	$\hat{k}=2k$	$\hat{k}=4k$
$\varepsilon=1$	precision	0.7 ± 0.1	0.6 ± 0.1	1.2 ± 0.4	1.6 ± 0.6	1.5 ± 0.5
	recall	0.2 ± 0.0	0.3 ± 0.1	1.3 ± 0.4	3.4 ± 1.3	6.4 ± 1.9
$\varepsilon=2$	precision	1.1 ± 1.5	2.1 ± 1.3	2.3 ± 1.2	2.1 ± 0.7	2.1 ± 0.6
	recall	0.3 ± 0.4	1.2 ± 0.7	2.5 ± 1.3	4.5 ± 1.6	9.2 ± 2.2
$\varepsilon=3$	precision	5.2 ± 4.3	5.8 ± 2.6	7.2 ± 1.8	6.4 ± 1.7	6.4 ± 1.5
	recall	1.4 ± 1.1	3.1 ± 1.3	7.8 ± 1.7	13.8 ± 3.1	27.8 ± 5.3
$\varepsilon=4$	precision	5.5 ± 2.7	7.5 ± 2.2	9.0 ± 2.0	11.6 ± 2.0	15.4 ± 2.3
	recall	1.5 ± 0.7	4.0 ± 1.0	9.7 ± 1.7	25.2 ± 3.6	66.9 ± 7.4
$\varepsilon=5$	precision	9.2 ± 3.9	12.4 ± 3.1	21.1 ± 3.2	30.9 ± 3.6	20.3 ± 1.1
	recall	2.5 ± 1.0	6.7 ± 1.4	22.9 ± 2.8	67.2 ± 5.3	88.6 ± 1.2
$\varepsilon=6$	precision	22.1 ± 4.5	36.2 ± 4.4	45.1 ± 2.8	36.5 ± 1.5	21.2 ± 0.9
	recall	6.0 ± 0.9	19.7 ± 1.6	49.2 ± 1.0	79.7 ± 1.6	92.5 ± 1.0
$\varepsilon=7$	precision	55.1 ± 2.4	61.9 ± 2.9	55.7 ± 3.2	39.8 ± 1.5	22.3 ± 0.8
	recall	15.0 ± 0.4	33.8 ± 0.4	60.8 ± 1.5	86.8 ± 0.6	97.1 ± 0.6
$\varepsilon=8$	precision	72.1 ± 6.4	75.6 ± 5.0	66.5 ± 3.1	43.2 ± 1.9	22.5 ± 1.0
	recall	19.7 ± 1.2	41.3 ± 1.5	72.6 ± 1.2	94.2 ± 0.2	98.3 ± 0.1
$\varepsilon=9$	precision	87.5 ± 1.9	85.9 ± 1.7	70.5 ± 2.2	43.5 ± 1.7	22.5 ± 1.0
	recall	23.9 ± 0.8	47.0 ± 1.1	77.0 ± 2.2	94.9 ± 1.4	98.2 ± 0.6
$\varepsilon=10$	precision	91.9 ± 3.3	86.9 ± 0.9	70.8 ± 2.3	43.0 ± 2.0	22.5 ± 1.1
	recall	25.1 ± 1.2	47.5 ± 1.6	77.3 ± 2.0	93.7 ± 1.5	98.0 ± 0.7

high		$\hat{k}=k/4$	$\hat{k}=k/2$	$\hat{k}=k$	$\hat{k}=2k$	$\hat{k}=4k$
$\varepsilon=1$	precision	15.6 ± 9.1	11.2 ± 4.6	9.2 ± 3.7	7.1 ± 2.5	5.5 ± 1.4
	recall	2.8 ± 1.4	4.0 ± 1.4	6.6 ± 2.2	10.2 ± 3.0	15.7 ± 3.0
$\varepsilon=2$	precision	22.3 ± 11.9	15.5 ± 5.8	12.4 ± 3.0	11.3 ± 2.2	10.0 ± 1.5
	recall	4.0 ± 1.9	5.6 ± 1.7	9.0 ± 1.6	16.4 ± 2.1	28.9 ± 2.5
$\varepsilon=3$	precision	27.5 ± 7.7	25.3 ± 4.9	24.6 ± 5.4	22.6 ± 3.2	20.3 ± 2.3
	recall	5.0 ± 1.1	9.2 ± 1.3	17.8 ± 2.7	32.9 ± 2.6	59.2 ± 3.7
$\varepsilon=4$	precision	38.7 ± 8.9	36.9 ± 5.9	37.6 ± 4.5	39.4 ± 3.4	29.8 ± 1.9
	recall	7.0 ± 1.3	13.5 ± 1.8	27.4 ± 1.8	57.4 ± 1.3	86.9 ± 2.2
$\varepsilon=5$	precision	55.9 ± 5.4	57.5 ± 3.3	59.6 ± 2.5	56.1 ± 2.9	32.2 ± 2.4
	recall	10.2 ± 0.7	21.0 ± 0.7	43.6 ± 1.7	81.9 ± 2.9	93.8 ± 1.2
$\varepsilon=6$	precision	71.9 ± 2.9	76.8 ± 1.6	78.1 ± 2.2	59.1 ± 4.2	32.7 ± 2.3
	recall	13.1 ± 0.6	28.1 ± 1.5	57.1 ± 2.6	86.2 ± 1.5	95.4 ± 0.8
$\varepsilon=7$	precision	88.1 ± 2.1	88.7 ± 1.5	86.0 ± 2.1	61.9 ± 3.7	33.5 ± 2.3
	recall	16.1 ± 0.8	32.5 ± 1.8	63.0 ± 3.2	90.3 ± 1.2	97.6 ± 0.5
$\varepsilon=8$	precision	94.0 ± 0.8	93.5 ± 0.5	88.4 ± 1.7	63.4 ± 3.5	33.8 ± 2.2
	recall	17.2 ± 1.2	34.3 ± 2.3	64.7 ± 3.4	92.6 ± 1.7	98.5 ± 0.6
$\varepsilon=9$	precision	96.8 ± 0.3	95.6 ± 0.8	89.0 ± 1.9	63.3 ± 3.8	33.7 ± 2.3
	recall	17.7 ± 1.3	35.0 ± 2.4	65.2 ± 3.3	92.4 ± 1.1	98.4 ± 0.5
$\varepsilon=10$	precision	97.5 ± 0.4	96.3 ± 0.3	89.4 ± 1.7	62.8 ± 4.2	33.5 ± 2.3
	recall	17.9 ± 1.3	35.3 ± 2.5	65.5 ± 3.7	91.6 ± 1.3	97.8 ± 0.4

TABLE XIV: twitch-RU (LAPGRAPH)

low		$\hat{k}=k/4$	$\hat{k}=k/2$	$\hat{k}=k$	$\hat{k}=2k$	$\hat{k}=4k$
$\varepsilon=1$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon=2$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon=3$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon=4$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.3 ± 0.5
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1.8 ± 2.5
$\varepsilon=5$	precision	0.0 ± 0.0	0.0 ± 0.0	5.3 ± 5.0	4.0 ± 3.3	2.3 ± 2.1
	recall	0.0 ± 0.0	0.0 ± 0.0	5.9 ± 5.4	8.8 ± 7.1	10.3 ± 8.9
$\varepsilon=6$	precision	0.0 ± 0.0	7.7 ± 6.3	6.7 ± 1.9	5.3 ± 2.5	4.3 ± 1.7
	recall	0.0 ± 0.0	4.5 ± 3.7	7.7 ± 1.7	12.2 ± 5.3	19.8 ± 6.7
$\varepsilon=7$	precision	23.8 ± 24.3	20.5 ± 13.1	14.7 ± 5.0	12.0 ± 3.3	9.7 ± 1.2
	recall	7.8 ± 7.6	12.2 ± 7.6	16.9 ± 5.0	27.8 ± 6.4	45.1 ± 2.3
$\varepsilon=8$	precision	28.6 ± 20.2	25.6 ± 7.3	21.3 ± 3.8	21.3 ± 4.1	16.3 ± 3.1
	recall	9.3 ± 6.3	15.4 ± 3.4	24.8 ± 2.7	49.7 ± 7.1	75.9 ± 9.6
$\varepsilon=9$	precision	42.9 ± 20.2	30.8 ± 12.6	32.0 ± 8.6	28.7 ± 5.0	18.0 ± 2.9
	recall	14.0 ± 6.2	18.4 ± 6.9	36.9 ± 7.6	66.8 ± 8.0	83.9 ± 8.9
$\varepsilon=10$	precision	52.4 ± 13.5	48.7 ± 3.6	48.0 ± 6.5	36.7 ± 2.5	19.7 ± 1.7
	recall	17.2 ± 4.1	29.8 ± 2.6	56.1 ± 5.3	86.2 ± 5.7	92.4 ± 7.3

unconstrained		$\hat{k}=k/4$	$\hat{k}=k/2$	$\hat{k}=k$	$\hat{k}=2k$	$\hat{k}=4k$
$\varepsilon=1$	precision	1.8 ± 0.7	1.8 ± 0.4	1.2 ± 0.2	0.9 ± 0.1	0.7 ± 0.1
	recall	0.5 ± 0.2	1.0 ± 0.2	1.3 ± 0.2	2.0 ± 0.3	2.9 ± 0.3
$\varepsilon=2$	precision	3.9 ± 0.1	3.1 ± 0.4	2.4 ± 0.7	1.4 ± 0.4	1.0 ± 0.1
	recall	1.1 ± 0.1	1.7 ± 0.3	2.6 ± 0.9	3.1 ± 1.1	4.5 ± 0.4
$\varepsilon=3$	precision	6.4 ± 1.6	7.6 ± 1.3	5.9 ± 0.7	3.4 ± 0.5	2.1 ± 0.3
	recall	1.7 ± 0.4	4.2 ± 0.7	6.5 ± 0.9	7.5 ± 1.4	9.1 ± 1.5
$\varepsilon=4$	precision	12.4 ± 2.6	13.2 ± 1.6	13.1 ± 1.5	7.6 ± 0.8	4.3 ± 0.3
	recall	3.4 ± 0.6	7.2 ± 0.7	14.3 ± 1.9	16.6 ± 2.1	19.0 ± 1.8
$\varepsilon=5$	precision	18.3 ± 0.8	26.9 ± 2.0	24.2 ± 0.8	14.8 ± 1.1	8.3 ± 0.3
	recall	5.0 ± 0.2	14.7 ± 0.5	26.5 ± 1.7	32.5 ± 3.5	36.3 ± 2.2
$\varepsilon=6$	precision	35.5 ± 3.4	43.0 ± 3.4	39.1 ± 1.3	25.5 ± 1.0	14.4 ± 0.4
	recall	9.7 ± 0.7	23.5 ± 1.1	42.6 ± 0.9	55.6 ± 2.6	62.9 ± 1.8
$\varepsilon=7$	precision	50.8 ± 8.8	57.3 ± 4.9	50.5 ± 2.4	32.3 ± 1.9	17.8 ± 0.8
	recall	13.8 ± 2.1	31.2 ± 1.9	55.1 ± 1.5	70.6 ± 1.4	77.8 ± 0.9
$\varepsilon=8$	precision	64.7 ± 7.6	65.8 ± 3.8	57.1 ± 2.3	36.5 ± 1.5	19.7 ± 0.8
	recall	17.7 ± 1.9	35.9 ± 0.6	62.3 ± 0.2	79.7 ± 0.1	85.8 ± 0.2
$\varepsilon=9$	precision	71.9 ± 4.0	72.7 ± 3.0	61.6 ± 3.1	39.3 ± 1.8	20.7 ± 0.8
	recall	19.6 ± 0.9	39.7 ± 0.3	67.3 ± 1.8	85.7 ± 0.5	90.4 ± 0.5
$\varepsilon=10$	precision	77.7 ± 3.5	78.4 ± 2.5	65.8 ± 2.5	40.6 ± 1.3	21.5 ± 0.7
	recall	21.3 ± 1.1	42.8 ± 1.3	71.8 ± 1.0	88.7 ± 1.0	93.7 ± 1.2

high		$\hat{k}=k/4$	$\hat{k}=k/2$	$\hat{k}=k$	$\hat{k}=2k$	$\hat{k}=4k$
$\varepsilon=1$	precision	3.6 ± 0.5	3.3 ± 1.1	2.5 ± 0.7	2.0 ± 0.3	1.8 ± 0.2
	recall	0.7 ± 0.1	1.2 ± 0.3	1.8 ± 0.4	2.9 ± 0.2	5.3 ± 0.3
$\varepsilon=2$	precision	10.5 ± 1.2	8.5 ± 1.9	5.3 ± 1.1	3.6 ± 0.6	2.5 ± 0.3
	recall	1.9 ± 0.2	3.1 ± 0.5	3.9 ± 0.6	5.2 ± 0.5	7.4 ± 0.6
$\varepsilon=3$	precision	22.2 ± 3.5	19.6 ± 2.0	11.5 ± 1.6	6.9 ± 1.1	4.3 ± 0.6
	recall	4.1 ± 0.6	7.1 ± 0.7	8.3 ± 0.9	10.1 ± 1.1	12.7 ± 1.2
$\varepsilon=4$	precision	39.3 ± 3.9	38.4 ± 3.4	23.6 ± 3.2	13.3 ± 1.9	8.3 ± 1.1
	recall	7.2 ± 0.4	14.0 ± 0.7	17.2 ± 1.3	19.4 ± 1.6	24.0 ± 1.7
$\varepsilon=5$	precision	56.9 ± 3.5	61.2 ± 2.3	44.2 ± 3.9	25.3 ± 2.3	14.5 ± 1.4
	recall	10.4 ± 0.4	22.4 ± 0.8	32.2 ± 0.9	36.8 ± 1.0	42.2 ± 1.3
$\varepsilon=6$	precision	73.8 ± 1.6	76.3 ± 1.1	66.9 ± 2.3	40.4 ± 2.9	22.2 ± 1.6
	recall	13.5 ± 0.9	28.0 ± 1.7	48.9 ± 1.8	58.9 ± 0.5	64.6 ± 0.0
$\varepsilon=7$	precision	83.9 ± 1.0	85.1 ± 1.2	76.9 ± 2.9	50.2 ± 3.5	27.0 ± 2.0
	recall	15.4 ± 1.1	31.2 ± 1.8	56.2 ± 1.9	73.2 ± 0.6	78.7 ± 0.2
$\varepsilon=8$	precision	90.2 ± 0.7	90.2 ± 0.2	82.5 ± 2.3	54.7 ± 3.3	29.8 ± 2.0
	recall	16.5 ± 1.1	33.1 ± 2.3	60.4 ± 2.7	79.9 ± 0.9	86.8 ± 0.3
$\varepsilon=9$	precision	93.3 ± 0.3	93.0 ± 0.5	85.8 ± 1.7	58.8 ± 3.6	31.5 ± 2.1
	recall	17.1 ± 1.2	34.1 ± 2.5	62.8 ± 3.4	85.8 ± 0.8	91.8 ± 0.3
$\varepsilon=10$	precision	95.1 ± 0.5	95.1 ± 0.2	88.1 ± 1.5	61.2 ± 4.1	32.5 ± 2.4
	recall	17.4 ± 1.2	34.9 ± 2.5	64.5 ± 3.6	89.3 ± 0.6	94.9 ± 0.2

TABLE XV: twitch-DE (EDGERAND)

low		$\hat{k} = k/4$	$\hat{k} = k/2$	$\hat{k} = k$	$\hat{k} = 2k$	$\hat{k} = 4k$
$\varepsilon = 1$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon = 2$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon = 3$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon = 4$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon = 5$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon = 6$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	2.6 ± 3.6	1.3 ± 1.9
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	5.6 ± 7.9	5.6 ± 7.9
$\varepsilon = 7$	precision	16.7 ± 23.6	8.3 ± 11.8	4.2 ± 5.9	2.2 ± 3.1	4.4 ± 4.2
	recall	4.8 ± 6.7	4.8 ± 6.7	4.8 ± 6.7	4.8 ± 6.7	18.7 ± 17.3
$\varepsilon = 8$	precision	25.0 ± 20.4	17.9 ± 12.7	21.7 ± 11.8	23.2 ± 13.6	16.7 ± 5.7
	recall	7.5 ± 5.9	10.3 ± 7.4	24.2 ± 12.4	49.2 ± 28.2	69.8 ± 23.4
$\varepsilon = 9$	precision	41.7 ± 31.2	31.0 ± 8.4	35.8 ± 12.8	32.9 ± 11.5	22.7 ± 1.9
	recall	13.1 ± 10.2	18.7 ± 4.6	40.1 ± 13.1	69.8 ± 23.4	95.2 ± 6.7
$\varepsilon = 10$	precision	83.3 ± 23.6	66.7 ± 23.6	54.0 ± 11.1	44.4 ± 4.2	23.8 ± 0.3
	recall	26.2 ± 7.0	40.1 ± 13.1	60.7 ± 10.5	94.4 ± 7.9	100.0 ± 0.0

unconstrained		$\hat{k} = k/4$	$\hat{k} = k/2$	$\hat{k} = k$	$\hat{k} = 2k$	$\hat{k} = 4k$
$\varepsilon = 1$	precision	3.9 ± 4.8	2.3 ± 2.2	1.4 ± 1.0	1.2 ± 0.4	1.1 ± 0.1
	recall	1.0 ± 1.2	1.2 ± 1.2	1.4 ± 1.0	2.3 ± 0.8	4.3 ± 0.6
$\varepsilon = 2$	precision	5.7 ± 6.6	3.7 ± 3.6	3.2 ± 0.9	2.8 ± 0.5	2.7 ± 0.6
	recall	1.4 ± 1.7	1.8 ± 1.9	3.1 ± 1.1	5.4 ± 0.7	10.2 ± 1.7
$\varepsilon = 3$	precision	6.4 ± 6.3	5.1 ± 3.5	5.3 ± 2.0	5.6 ± 1.6	6.0 ± 1.6
	recall	1.6 ± 1.6	2.5 ± 1.8	5.1 ± 2.1	10.5 ± 2.6	22.5 ± 4.7
$\varepsilon = 4$	precision	8.9 ± 5.3	9.0 ± 4.6	10.1 ± 2.0	9.4 ± 0.9	11.6 ± 1.5
	recall	2.2 ± 1.4	4.4 ± 2.5	9.7 ± 2.3	17.9 ± 0.8	43.9 ± 2.5
$\varepsilon = 5$	precision	12.4 ± 4.9	16.5 ± 2.0	19.4 ± 2.7	23.3 ± 1.1	23.3 ± 2.6
	recall	3.0 ± 1.4	7.9 ± 1.2	18.5 ± 2.3	44.4 ± 1.5	88.1 ± 3.9
$\varepsilon = 6$	precision	27.7 ± 4.6	33.7 ± 1.5	40.4 ± 4.9	42.2 ± 5.1	24.6 ± 2.3
	recall	6.6 ± 1.2	16.1 ± 0.4	38.3 ± 2.5	79.9 ± 4.8	93.2 ± 2.4
$\varepsilon = 7$	precision	55.7 ± 8.6	60.1 ± 5.8	61.8 ± 7.4	46.6 ± 4.6	25.4 ± 2.3
	recall	13.3 ± 2.3	28.7 ± 1.6	58.6 ± 2.9	88.3 ± 2.9	96.4 ± 1.9
$\varepsilon = 8$	precision	73.0 ± 3.5	77.1 ± 2.4	71.7 ± 6.5	48.1 ± 4.3	25.7 ± 2.2
	recall	17.5 ± 1.8	36.9 ± 1.5	68.2 ± 2.9	91.3 ± 2.2	97.5 ± 1.8
$\varepsilon = 9$	precision	84.8 ± 1.8	84.6 ± 4.4	77.2 ± 6.7	48.5 ± 4.0	25.5 ± 1.7
	recall	20.3 ± 1.0	40.4 ± 1.1	73.5 ± 4.5	92.1 ± 1.6	97.0 ± 0.3
$\varepsilon = 10$	precision	88.3 ± 1.5	90.2 ± 3.8	80.2 ± 5.6	49.1 ± 4.6	25.6 ± 1.9
	recall	21.1 ± 1.1	43.2 ± 1.5	76.4 ± 3.2	93.2 ± 2.3	97.3 ± 0.6

high		$\hat{k} = k/4$	$\hat{k} = k/2$	$\hat{k} = k$	$\hat{k} = 2k$	$\hat{k} = 4k$
$\varepsilon = 1$	precision	3.6 ± 2.4	3.1 ± 0.9	2.9 ± 0.7	2.4 ± 0.2	2.5 ± 0.4
	recall	0.9 ± 0.6	1.6 ± 0.5	3.0 ± 0.8	4.9 ± 0.5	10.3 ± 1.8
$\varepsilon = 2$	precision	7.2 ± 2.0	6.0 ± 1.1	6.0 ± 0.5	5.9 ± 0.7	5.6 ± 0.7
	recall	1.8 ± 0.5	3.1 ± 0.7	6.1 ± 0.6	12.1 ± 1.9	22.8 ± 3.8
$\varepsilon = 3$	precision	12.9 ± 0.8	11.6 ± 1.4	12.4 ± 0.4	13.0 ± 0.1	12.4 ± 0.3
	recall	3.3 ± 0.3	5.9 ± 1.0	12.6 ± 0.8	26.4 ± 1.1	50.3 ± 3.8
$\varepsilon = 4$	precision	20.2 ± 2.2	21.0 ± 0.9	23.3 ± 0.9	23.9 ± 0.8	20.3 ± 0.5
	recall	5.2 ± 0.7	10.7 ± 1.0	23.7 ± 2.0	48.6 ± 3.9	82.3 ± 2.7
$\varepsilon = 5$	precision	34.6 ± 1.4	38.6 ± 1.8	43.1 ± 1.9	42.0 ± 0.9	23.1 ± 0.9
	recall	8.8 ± 0.7	19.7 ± 1.9	43.8 ± 3.9	85.3 ± 3.8	93.6 ± 1.6
$\varepsilon = 6$	precision	52.5 ± 1.9	59.5 ± 0.9	68.0 ± 0.9	44.7 ± 2.0	24.0 ± 1.2
	recall	13.4 ± 0.9	30.2 ± 1.6	69.0 ± 4.2	90.7 ± 2.0	97.1 ± 0.6
$\varepsilon = 7$	precision	75.5 ± 3.3	80.0 ± 1.6	75.9 ± 0.9	45.8 ± 1.7	24.2 ± 1.2
	recall	19.2 ± 1.8	40.6 ± 2.7	77.1 ± 3.6	92.8 ± 1.2	97.9 ± 0.1
$\varepsilon = 8$	precision	89.1 ± 3.3	90.1 ± 1.9	77.9 ± 0.3	46.3 ± 1.4	24.2 ± 1.0
	recall	22.7 ± 1.9	45.8 ± 3.2	79.1 ± 3.6	93.8 ± 2.0	98.0 ± 0.9
$\varepsilon = 9$	precision	95.1 ± 0.3	93.0 ± 1.3	78.4 ± 0.7	46.3 ± 2.0	24.0 ± 1.2
	recall	24.2 ± 1.1	47.2 ± 2.9	79.5 ± 3.5	93.8 ± 0.8	97.0 ± 0.1
$\varepsilon = 10$	precision	97.7 ± 1.0	95.6 ± 0.3	79.1 ± 1.4	46.6 ± 1.8	24.3 ± 1.2
	recall	24.8 ± 1.0	48.5 ± 2.5	80.3 ± 2.9	94.4 ± 1.2	98.3 ± 0.4

TABLE XVI: twitch-DE (LAPGRAPH)

low		$\hat{k} = k/4$	$\hat{k} = k/2$	$\hat{k} = k$	$\hat{k} = 2k$	$\hat{k} = 4k$
$\varepsilon = 1$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon = 2$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon = 3$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon = 4$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon = 5$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.7 ± 0.9
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	2.8 ± 3.9
$\varepsilon = 6$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	2.7 ± 1.9
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	11.1 ± 7.9
$\varepsilon = 7$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	2.2 ± 3.1	6.2 ± 1.7
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	4.8 ± 6.7	26.2 ± 7.0
$\varepsilon = 8$	precision	0.0 ± 0.0	0.0 ± 0.0	5.1 ± 7.3	8.8 ± 2.3	7.3 ± 2.5
	recall	0.0 ± 0.0	0.0 ± 0.0	5.6 ± 7.9	18.7 ± 4.6	31.0 ± 10.8
$\varepsilon = 9$	precision	8.3 ± 11.8	4.8 ± 6.7	5.1 ± 7.3	13.7 ± 5.0	11.8 ± 2.7
	recall	2.8 ± 3.9	2.8 ± 3.9	5.6 ± 7.9	29.0 ± 10.2	49.6 ± 11.5
$\varepsilon = 10$	precision	0.0 ± 0.0	8.3 ± 11.8	23.9 ± 8.1	21.1 ± 5.2	18.2 ± 2.7
	recall	0.0 ± 0.0	5.6 ± 7.9	27.0 ± 9.0	44.8 ± 10.3	76.6 ± 10.8

unconstrained		$\hat{k} = k/4$	$\hat{k} = k/2$	$\hat{k} = k$	$\hat{k} = 2k$	$\hat{k} = 4k$
$\varepsilon = 1$	precision	1.1 ± 0.9	0.7 ± 0.3	0.9 ± 0.3	0.8 ± 0.5	0.6 ± 0.3
	recall	0.3 ± 0.2	0.3 ± 0.1	0.9 ± 0.4	1.5 ± 0.8	2.3 ± 0.9
$\varepsilon = 2$	precision	2.5 ± 1.3	2.0 ± 0.9	1.9 ± 0.8	1.4 ± 0.7	1.0 ± 0.5
	recall	0.6 ± 0.3	0.9 ± 0.4	1.7 ± 0.6	2.6 ± 1.1	3.7 ± 1.6
$\varepsilon = 3$	precision	6.4 ± 1.7	5.7 ± 1.3	4.8 ± 0.6	3.0 ± 0.5	1.8 ± 0.4
	recall	1.5 ± 0.3	2.7 ± 0.4	4.6 ± 0.2	5.7 ± 0.6	7.0 ± 1.3
$\varepsilon = 4$	precision	11.0 ± 1.8	12.2 ± 2.6	12.3 ± 1.5	7.3 ± 0.8	4.0 ± 0.6
	recall	2.6 ± 0.4	5.8 ± 0.9	11.6 ± 0.6	13.8 ± 0.7	15.1 ± 1.3
$\varepsilon = 5$	precision	17.4 ± 3.3	21.5 ± 3.7	26.7 ± 3.1	16.5 ± 1.9	8.6 ± 0.9
	recall	4.2 ± 0.9	10.2 ± 1.6	25.3 ± 1.6	31.4 ± 2.4	32.5 ± 2.1
$\varepsilon = 6$	precision	30.1 ± 4.8	36.0 ± 3.7	40.1 ± 3.3	26.4 ± 1.9	14.2 ± 1.2
	recall	7.2 ± 1.2	17.2 ± 1.1	38.2 ± 2.0	50.2 ± 2.2	54.0 ± 2.3
$\varepsilon = 7$	precision	41.8 ± 5.7	50.4 ± 5.1	54.0 ± 3.6	35.2 ± 2.5	19.4 ± 1.8
	recall	10.0 ± 1.0	24.0 ± 1.2	51.4 ± 0.3	67.0 ± 1.5	73.8 ± 2.7
$\varepsilon = 8$	precision	54.3 ± 3.8	61.5 ± 1.8	61.2 ± 4.1	40.2 ± 2.6	21.9 ± 1.2
	recall	13.0 ± 1.3	29.5 ± 1.9	58.2 ± 0.6	76.5 ± 0.7	83.3 ± 1.8
$\varepsilon = 9$	precision	66.3 ± 2.7	72.3 ± 4.6	68.1 ± 6.6	44.2 ± 3.6	23.8 ± 1.6
	recall	15.9 ± 1.0	34.5 ± 0.4	64.7 ± 2.0	84.0 ± 1.6	90.4 ± 1.2
$\varepsilon = 10$	precision	73.8 ± 3.9	78.0 ± 3.7	73.4 ± 7.2	46.8 ± 3.4	24.6 ± 1.6
	recall	17.6 ± 0.3	37.3 ± 0.8	69.8 ± 2.4	89.0 ± 0.5	93.4 ± 1.1

high		$\hat{k} = k/4$	$\hat{k} = k/2$	$\hat{k} = k$	$\hat{k} = 2k$	$\hat{k} = 4k$
$\varepsilon = 1$	precision	2.1 ± 1.0	1.9 ± 0.6	1.6 ± 0.5	1.3 ± 0.3	1.1 ± 0.1
	recall	0.5 ± 0.2	1.0 ± 0.3	1.6 ± 0.4	2.6 ± 0.6	4.5 ± 0.4
$\varepsilon = 2$	precision	5.5 ± 1.3	5.2 ± 0.4	3.2 ± 0.2	2.2 ± 0.0	1.6 ± 0.0
	recall	1.4 ± 0.4	2.7 ± 0.2	3.2 ± 0.1	4.4 ± 0.2	6.3 ± 0.4
$\varepsilon = 3$	precision	13.7 ± 1.4	12.7 ± 0.3	7.6 ± 0.3	4.3 ± 0.3	2.6 ± 0.2
	recall	3.5 ± 0.5	6.5 ± 0.5	7.8 ± 0.6	8.7 ± 0.9	10.6 ± 1.2
$\varepsilon = 4$	precision	26.1 ± 1.2	26.2 ± 0.7	15.4 ± 1.1	8.5 ± 0.5	4.8 ± 0.2
	recall	6.6 ± 0.6	13.3 ± 0.5	15.6 ± 0.4	17.2 ± 0.3	19.5 ± 0.4
$\varepsilon = 5$	precision	38.9 ± 2.0	47.1 ± 1.7	31.4 ± 1.8	17.3 ± 1.2	9.5 ± 0.4
	recall	9.9 ± 0.9	24.0 ± 2.0	31.8 ± 0.2	34.9 ± 0.7	38.6 ± 0.8
$\varepsilon = 6$	precision	58.1 ± 3.1	66.4 ± 2.9	50.1 ± 2.2	27.9 ± 1.3	15.2 ± 0.7
	recall	14.8 ± 1.4	33.8 ± 2.9	50.8 ± 2.0	56.6 ± 1.7	61.7 ± 2.7
$\varepsilon = 7$	precision	73.9 ± 1.7	77.6 ± 0.5	63.4 ± 2.0	36.2 ± 1.7	19.2 ± 0.8
	recall	18.8 ± 1.1	39.4 ± 2.2	64.4 ± 2.4	73.3 ± 1.4	77.7 ± 1.7
$\varepsilon = 8$	precision	84.5 ± 1.0	85.2 ± 1.3	68.0 ± 1.5	39.9 ± 1.0	21.0 ± 0.7
	recall	21.5 ± 1.3	43.3 ± 2.7	69.0 ± 2.5	80.9 ± 2.1	85.1 ± 1.7
$\varepsilon = 9$	precision	90.7 ± 1.8	88.9 ± 0.7	73.7 ± 1.2	42.6 ± 1.4	22.4 ± 0.7
	recall	23.1 ± 1.5	45.1 ± 2.3	74.7 ± 2.5	86.4 ± 1.6	90.6 ± 1.7
$\varepsilon = 10$	precision	94.0 ± 0.3	92.6 ± 0.7	76.1 ± 1.4	44.4 ± 1.9	23.2 ± 1.1
	recall	23.9 ± 1.2	47.0 ± 2.5	77.2 ± 2.7	90.0 ± 0.8	94.1 ± 0.6

TABLE XVII: twitch-FR (EDGERAND)

low		$\hat{k}=k/4$	$\hat{k}=k/2$	$\hat{k}=k$	$\hat{k}=2k$	$\hat{k}=4k$
$\varepsilon=1$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon=2$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon=3$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon=4$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon=5$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon=6$	precision	0.0 ± 0.0	6.7 ± 9.4	3.3 ± 4.7	1.7 ± 2.4	4.4 ± 2.2
	recall	0.0 ± 0.0	3.3 ± 4.7	3.3 ± 4.7	3.3 ± 4.7	17.5 ± 8.9
$\varepsilon=7$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	5.0 ± 4.1	8.6 ± 3.9
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	10.0 ± 8.2	34.2 ± 15.9
$\varepsilon=8$	precision	0.0 ± 0.0	6.7 ± 9.4	6.7 ± 9.4	15.6 ± 6.3	19.2 ± 1.2
	recall	0.0 ± 0.0	3.3 ± 4.7	6.7 ± 9.4	30.8 ± 13.0	75.0 ± 4.1
$\varepsilon=9$	precision	38.9 ± 7.9	28.3 ± 8.5	32.5 ± 3.5	35.6 ± 7.5	25.6 ± 0.8
	recall	10.8 ± 1.2	14.2 ± 4.2	32.5 ± 3.5	70.0 ± 16.3	100.0± 0.0
$\varepsilon=10$	precision	77.8 ± 15.7	80.0 ± 16.3	71.7 ± 2.4	49.4 ± 3.4	25.6 ± 0.8
	recall	21.7 ± 2.4	40.0 ± 8.2	71.7 ± 2.4	96.7 ± 4.7	100.0± 0.0

unconstrained		$\hat{k}=k/4$	$\hat{k}=k/2$	$\hat{k}=k$	$\hat{k}=2k$	$\hat{k}=4k$
$\varepsilon=1$	precision	7.1 ± 7.2	3.8 ± 3.4	2.5 ± 2.0	1.8 ± 1.0	1.5 ± 0.6
	recall	1.7 ± 1.8	1.8 ± 1.7	2.4 ± 1.9	3.4 ± 2.1	5.5 ± 2.6
$\varepsilon=2$	precision	9.1 ± 9.4	5.4 ± 4.8	4.5 ± 3.1	3.7 ± 1.9	3.6 ± 1.3
	recall	2.2 ± 2.3	2.6 ± 2.4	4.3 ± 3.1	6.9 ± 3.9	13.5 ± 5.3
$\varepsilon=3$	precision	11.2 ± 10.3	9.1 ± 6.1	8.1 ± 4.2	9.1 ± 2.8	9.6 ± 2.7
	recall	2.7 ± 2.5	4.3 ± 3.0	7.7 ± 4.3	17.0 ± 5.7	35.9 ± 11.2
$\varepsilon=4$	precision	15.0 ± 12.7	16.7 ± 5.4	17.0 ± 4.9	18.7 ± 4.0	18.3 ± 2.8
	recall	3.6 ± 3.1	7.8 ± 2.8	16.0 ± 5.0	35.0 ± 8.2	68.2 ± 11.7
$\varepsilon=5$	precision	21.0 ± 10.1	23.9 ± 7.6	29.1 ± 6.5	36.0 ± 3.5	24.5 ± 0.8
	recall	5.0 ± 2.6	11.3 ± 3.9	27.3 ± 7.0	67.2 ± 8.6	91.0 ± 3.0
$\varepsilon=6$	precision	41.5 ± 6.2	47.9 ± 6.3	57.7 ± 4.6	46.4 ± 1.8	25.5 ± 0.9
	recall	9.7 ± 1.8	22.4 ± 3.7	53.8 ± 6.0	86.4 ± 3.5	94.9 ± 1.9
$\varepsilon=7$	precision	63.2 ± 4.1	68.1 ± 3.8	69.6 ± 3.8	48.9 ± 1.7	25.9 ± 0.7
	recall	14.8 ± 1.3	31.8 ± 2.8	64.8 ± 5.3	90.9 ± 2.5	96.4 ± 0.9
$\varepsilon=8$	precision	83.1 ± 3.9	84.0 ± 3.5	76.8 ± 3.2	49.8 ± 1.9	26.1 ± 0.9
	recall	19.4 ± 1.4	39.2 ± 2.5	71.5 ± 2.5	92.5 ± 1.2	97.2 ± 0.6
$\varepsilon=9$	precision	90.4 ± 1.2	91.5 ± 0.9	82.2 ± 3.9	51.0 ± 2.4	26.3 ± 1.0
	recall	21.1 ± 0.8	42.6 ± 1.1	76.5 ± 2.3	94.8 ± 1.7	97.8 ± 0.7
$\varepsilon=10$	precision	94.6 ± 0.3	92.6 ± 1.3	82.4 ± 2.7	51.1 ± 1.9	26.6 ± 0.9
	recall	22.1 ± 0.8	43.2 ± 1.3	76.7 ± 1.6	94.9 ± 0.9	98.8 ± 0.4

high		$\hat{k}=k/4$	$\hat{k}=k/2$	$\hat{k}=k$	$\hat{k}=2k$	$\hat{k}=4k$
$\varepsilon=1$	precision	4.6 ± 4.2	3.7 ± 2.4	3.4 ± 0.9	3.2 ± 1.0	3.2 ± 0.5
	recall	1.1 ± 1.1	1.8 ± 1.2	3.3 ± 1.0	6.4 ± 2.4	12.5 ± 2.9
$\varepsilon=2$	precision	8.7 ± 2.4	8.5 ± 2.5	8.5 ± 3.2	8.1 ± 2.4	7.7 ± 1.2
	recall	2.1 ± 0.7	4.2 ± 1.5	8.5 ± 3.7	15.9 ± 5.9	30.1 ± 6.6
$\varepsilon=3$	precision	14.9 ± 1.6	18.3 ± 1.0	19.2 ± 0.2	18.2 ± 0.4	16.8 ± 0.5
	recall	3.6 ± 0.5	8.8 ± 0.5	18.6 ± 1.5	35.3 ± 2.8	65.0 ± 5.1
$\varepsilon=4$	precision	31.8 ± 3.0	32.6 ± 2.2	32.2 ± 0.4	32.9 ± 0.3	21.9 ± 0.5
	recall	7.8 ± 1.3	15.8 ± 1.9	31.2 ± 2.3	63.7 ± 5.6	84.7 ± 6.4
$\varepsilon=5$	precision	45.3 ± 2.5	49.6 ± 4.1	53.9 ± 1.1	44.2 ± 1.4	24.0 ± 1.1
	recall	11.0 ± 1.5	24.2 ± 3.8	52.3 ± 5.5	85.5 ± 5.1	92.6 ± 4.3
$\varepsilon=6$	precision	65.0 ± 1.6	71.7 ± 0.2	74.8 ± 1.9	46.8 ± 2.2	24.6 ± 1.3
	recall	15.7 ± 1.4	34.8 ± 2.9	72.5 ± 6.7	90.3 ± 3.7	95.0 ± 3.4
$\varepsilon=7$	precision	80.4 ± 2.6	84.5 ± 2.6	79.5 ± 1.1	48.6 ± 2.9	25.3 ± 1.9
	recall	19.5 ± 2.0	41.0 ± 3.9	77.0 ± 5.8	93.6 ± 2.6	97.4 ± 1.4
$\varepsilon=8$	precision	92.3 ± 2.3	92.8 ± 1.2	80.5 ± 1.3	48.4 ± 2.7	25.2 ± 1.7
	recall	22.3 ± 1.8	45.0 ± 3.7	77.9 ± 5.5	93.3 ± 3.1	97.0 ± 2.1
$\varepsilon=9$	precision	97.1 ± 1.2	96.0 ± 1.0	84.2 ± 1.4	49.0 ± 2.7	25.4 ± 1.9
	recall	23.5 ± 2.1	46.5 ± 4.2	81.6 ± 6.2	94.6 ± 3.1	97.8 ± 1.2
$\varepsilon=10$	precision	98.0 ± 0.8	97.2 ± 0.8	84.9 ± 1.1	49.2 ± 2.9	25.3 ± 1.8
	recall	23.7 ± 2.1	47.1 ± 4.3	82.2 ± 6.6	94.8 ± 2.9	97.5 ± 1.8

TABLE XVIII: twitch-FR (LAPGRAPH)

low		$\hat{k}=k/4$	$\hat{k}=k/2$	$\hat{k}=k$	$\hat{k}=2k$	$\hat{k}=4k$
$\varepsilon=1$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon=2$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1.1 ± 1.6
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	4.2 ± 5.9
$\varepsilon=3$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	2.2 ± 3.1	1.9 ± 1.4
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	4.2 ± 5.9	7.5 ± 5.4
$\varepsilon=4$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1.1 ± 1.6
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	4.2 ± 5.9
$\varepsilon=5$	precision	0.0 ± 0.0	0.0 ± 0.0	3.3 ± 4.7	3.9 ± 2.8	1.9 ± 1.4
	recall	0.0 ± 0.0	0.0 ± 0.0	3.3 ± 4.7	7.5 ± 5.4	7.5 ± 5.4
$\varepsilon=6$	precision	11.1 ± 15.7	6.7 ± 9.4	3.3 ± 4.7	5.6 ± 4.2	3.6 ± 1.0
	recall	3.3 ± 4.7	3.3 ± 4.7	3.3 ± 4.7	10.8 ± 8.2	14.2 ± 4.2
$\varepsilon=7$	precision	22.2 ± 31.4	13.3 ± 18.9	10.8 ± 8.2	5.6 ± 4.2	5.3 ± 1.7
	recall	6.7 ± 9.4	6.7 ± 9.4	10.8 ± 8.2	10.8 ± 8.2	20.8 ± 7.2
$\varepsilon=8$	precision	0.0 ± 0.0	6.7 ± 9.4	13.3 ± 12.5	8.3 ± 6.2	9.7 ± 2.4
	recall	0.0 ± 0.0	3.3 ± 4.7	13.3 ± 12.5	16.7 ± 12.5	38.3 ± 10.3
$\varepsilon=9$	precision	11.1 ± 15.7	6.7 ± 9.4	6.7 ± 9.4	13.3 ± 10.3	15.8 ± 6.6
	recall	3.3 ± 4.7	3.3 ± 4.7	6.7 ± 9.4	26.7 ± 20.5	62.5 ± 27.0
$\varepsilon=10$	precision	0.0 ± 0.0	6.7 ± 9.4	16.7 ± 12.5	22.2 ± 11.7	20.6 ± 3.4
	recall	0.0 ± 0.0	3.3 ± 4.7	16.7 ± 12.5	44.2 ± 23.8	80.8 ± 15.3

unconstrained		$\hat{k}=k/4$	$\hat{k}=k/2$	$\hat{k}=k$	$\hat{k}=2k$	$\hat{k}=4k$
$\varepsilon=1$	precision	1.8 ± 0.3	2.1 ± 0.4	2.0 ± 0.4	1.4 ± 0.2	1.1 ± 0.2
	recall	0.4 ± 0.1	1.0 ± 0.2	1.9 ± 0.5	2.7 ± 0.6	4.0 ± 0.8
$\varepsilon=2$	precision	3.8 ± 0.8	4.3 ± 1.4	4.1 ± 0.9	2.7 ± 0.3	1.6 ± 0.3
	recall	0.9 ± 0.2	2.0 ± 0.7	3.8 ± 1.0	5.0 ± 0.6	6.0 ± 1.2
$\varepsilon=3$	precision	7.5 ± 0.8	7.7 ± 2.2	10.0 ± 0.8	6.1 ± 0.1	3.4 ± 0.2
	recall	1.8 ± 0.2	3.6 ± 1.2	9.3 ± 1.1	11.4 ± 0.4	12.7 ± 0.9
$\varepsilon=4$	precision	15.1 ± 5.6	17.9 ± 4.8	18.6 ± 2.8	12.2 ± 0.4	6.6 ± 0.2
	recall	3.6 ± 1.5	8.4 ± 2.5	17.4 ± 3.2	22.8 ± 1.3	24.5 ± 1.7
$\varepsilon=5$	precision	23.5 ± 9.0	29.0 ± 7.2	35.7 ± 3.1	22.9 ± 0.5	12.3 ± 0.3
	recall	5.5 ± 2.3	13.6 ± 3.8	33.3 ± 4.0	42.6 ± 1.4	45.7 ± 1.9
$\varepsilon=6$	precision	35.8 ± 8.4	47.0 ± 6.6	51.3 ± 4.1	33.4 ± 0.4	18.0 ± 0.2
	recall	8.4 ± 2.3	22.0 ± 3.8	47.8 ± 5.0	62.1 ± 1.6	67.2 ± 2.5
$\varepsilon=7$	precision	52.8 ± 6.5	62.3 ± 6.0	62.6 ± 3.2	40.1 ± 0.9	21.4 ± 0.6
	recall	12.4 ± 1.9	29.1 ± 3.6	58.4 ± 4.2	74.7 ± 1.8	79.7 ± 1.8
$\varepsilon=8$	precision	66.9 ± 5.9	73.3 ± 4.5	69.6 ± 2.8	43.8 ± 1.8	23.6 ± 0.7
	recall	15.7 ± 1.8	34.2 ± 3.1	64.8 ± 3.4	81.5 ± 1.5	87.6 ± 1.6
$\varepsilon=9$	precision	78.1 ± 4.2	81.1 ± 1.9	74.1 ± 1.2	46.9 ± 1.3	24.6 ± 0.8
	recall	18.3 ± 1.6	37.8 ± 2.1	69.0 ± 3.1	87.3 ± 2.2	91.6 ± 1.2
$\varepsilon=10$	precision	85.3 ± 1.4	85.9 ± 1.9	77.3 ± 2.0	48.2 ± 1.6	25.3 ± 0.7
	recall	19.9 ± 1.0	40.1 ± 2.1	72.0 ± 2.7	89.7 ± 2.2	94.0 ± 0.7

high		$\hat{k}=k/4$	$\hat{k}=k/2$	$\hat{k}=k$	$\hat{k}=2k$	$\hat{k}=4k$
$\varepsilon=1$	precision	3.0 ± 1.0	3.2 ± 1.2	2.7 ± 0.7	2.4 ± 0.3	1.7 ± 0.2
	recall	0.7 ± 0.2	1.5 ± 0.4	2.6 ± 0.4	4.5 ± 0.4	6.6 ± 0.2
$\varepsilon=2$	precision	7.8 ± 0.6	8.4 ± 0.9	5.3 ± 0.7	3.5 ± 0.6	2.6 ± 0.3
	recall	1.9 ± 0.0	4.1 ± 0.2	5.1 ± 0.2	6.7 ± 0.5	9.8 ± 0.5
$\varepsilon=3$	precision	16.3 ± 1.2	17.3 ± 0.8	10.8 ± 1.4	6.2 ± 0.8	3.9 ± 0.4
	recall	3.9 ± 0.3	8.3 ± 0.4	10.3 ± 0.4	11.8 ± 0.5	14.8 ± 0.2
$\varepsilon=4$	precision	30.2 ± 0.6	33.9 ± 1.4	22.9 ± 1.7	12.9 ± 1.3	7.4 ± 0.6
	recall	7.3 ± 0.7	16.4 ± 0.8	22.1 ± 0.6	24.7 ± 0.5	28.5 ± 0.4
$\varepsilon=5$	precision	47.0 ± 1.9	54.0 ± 2.4	42.6 ± 2.6	23.7 ± 2.1	13.0 ± 1.1
	recall	11.4 ± 1.1	26.2 ± 2.7	41.1 ± 1.2	45.5 ± 0.5	50.0 ± 1.4
$\varepsilon=6$	precision	67.9 ± 2.5	73.1 ± 0.9	60.0 ± 0.9	33.9 ± 1.8	18.4 ± 1.2
	recall	16.5 ± 1.8	35.4 ± 3.1	58.1 ± 4.1	65.4 ± 2.3	70.9 ± 1.7
$\varepsilon=7$	precision	79.8 ± 3.8	83.4 ± 1.7	71.4 ± 2.0	41.0 ± 3.2	21.6 ± 1.5
	recall	19.3 ± 1.9	40.4 ± 3.3	69.0 ± 4.1	78.9 ± 1.0	83.2 ± 1.5
$\varepsilon=8$	precision	87.7 ± 2.9	89.5 ± 1.6	74.7 ± 1.5	43.7 ± 2.5	22.8 ± 1.5
	recall	21.3 ± 2.0	43.4 ± 3.9	72.4 ± 5.5	84.2 ± 2.5	88.0 ± 2.1
$\varepsilon=9$	precision	91.5 ± 2.1	91.9 ± 1.4	78.0 ± 1.0	46.0 ± 2.4	23.8 ± 1.5
	recall	22.1 ± 1.8	44.6 ± 4.2	75.7 ± 7.3	88.7 ± 3.1	91.7 ± 2.3
$\varepsilon=10$	precision	95.0 ± 0.9	93.9 ± 1.5	80.2 ± 0.6	46.7 ± 2.5	24.4 ± 1.6
	recall	23.0 ± 2.0	45.5 ± 4.5	77.7 ± 6.0	90.1 ± 3.0	93.9 ± 2.1

TABLE XIX: twitch-ENGB (EDGERAND)

low		$\hat{k}=k/4$	$\hat{k}=k/2$	$\hat{k}=k$	$\hat{k}=2k$	$\hat{k}=4k$
$\varepsilon=1$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon=2$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon=3$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon=4$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.7 ± 0.9	2.3 ± 1.2
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1.7 ± 2.4	8.4 ± 3.1
$\varepsilon=5$	precision	8.3 ± 11.8	4.8 ± 6.7	2.6 ± 3.6	3.3 ± 0.9	5.0 ± 0.8
	recall	2.1 ± 2.9	2.1 ± 2.9	2.1 ± 2.9	6.3 ± 1.1	19.7 ± 4.0
$\varepsilon=6$	precision	13.1 ± 10.2	19.4 ± 6.5	18.2 ± 3.5	14.7 ± 1.9	9.7 ± 1.2
	recall	4.2 ± 3.2	10.5 ± 3.7	18.5 ± 4.4	29.8 ± 8.7	39.5 ± 12.3
$\varepsilon=7$	precision	13.1 ± 10.2	22.0 ± 5.9	18.4 ± 10.1	16.0 ± 3.3	14.0 ± 3.3
	recall	3.8 ± 2.7	11.7 ± 3.0	20.1 ± 14.2	32.7 ± 12.3	58.4 ± 24.2
$\varepsilon=8$	precision	56.0 ± 13.8	46.5 ± 12.8	37.7 ± 5.4	33.3 ± 3.8	20.7 ± 0.9
	recall	16.9 ± 4.5	26.1 ± 9.8	39.5 ± 12.3	67.7 ± 18.9	81.9 ± 13.2
$\varepsilon=9$	precision	60.7 ± 10.5	58.6 ± 2.1	54.6 ± 1.0	41.3 ± 1.9	26.0 ± 4.9
	recall	18.0 ± 1.9	31.9 ± 6.2	55.9 ± 10.8	81.9 ± 13.2	100.0± 0.0
$\varepsilon=10$	precision	83.3 ± 11.8	82.8 ± 4.1	74.1 ± 3.4	46.7 ± 6.8	26.0 ± 4.9
	recall	25.6 ± 6.9	44.6 ± 5.2	75.6 ± 14.0	90.7 ± 6.7	100.0± 0.0

unconstrained		$\hat{k}=k/4$	$\hat{k}=k/2$	$\hat{k}=k$	$\hat{k}=2k$	$\hat{k}=4k$
$\varepsilon=1$	precision	1.1 ± 1.5	0.8 ± 0.7	0.5 ± 0.4	0.6 ± 0.5	0.4 ± 0.4
	recall	0.4 ± 0.5	0.5 ± 0.4	0.7 ± 0.5	1.5 ± 1.2	2.2 ± 1.8
$\varepsilon=2$	precision	1.1 ± 1.5	1.3 ± 1.0	1.3 ± 0.8	0.9 ± 0.2	0.7 ± 0.2
	recall	0.3 ± 0.5	0.7 ± 0.6	1.3 ± 0.5	1.9 ± 0.4	3.3 ± 1.3
$\varepsilon=3$	precision	0.0 ± 0.0	0.8 ± 1.1	1.2 ± 0.9	1.5 ± 0.6	1.9 ± 0.8
	recall	0.0 ± 0.0	0.5 ± 0.7	1.5 ± 1.1	3.6 ± 1.8	8.7 ± 4.5
$\varepsilon=4$	precision	2.1 ± 2.0	1.6 ± 1.7	3.2 ± 1.3	3.3 ± 1.0	4.5 ± 1.1
	recall	0.7 ± 0.7	1.0 ± 1.1	3.8 ± 1.9	7.8 ± 3.4	20.7 ± 8.3
$\varepsilon=5$	precision	3.7 ± 0.8	4.8 ± 1.1	8.1 ± 0.7	11.2 ± 1.2	13.5 ± 3.0
	recall	1.1 ± 0.4	2.5 ± 0.1	9.1 ± 2.6	25.1 ± 7.9	61.8 ± 24.3
$\varepsilon=6$	precision	15.2 ± 2.6	16.5 ± 0.9	22.9 ± 3.2	27.2 ± 3.7	20.1 ± 3.8
	recall	4.1 ± 0.3	9.0 ± 1.7	25.9 ± 8.7	61.2 ± 19.9	84.3 ± 6.4
$\varepsilon=7$	precision	30.5 ± 3.0	38.3 ± 5.0	40.1 ± 4.7	34.3 ± 3.8	21.2 ± 4.2
	recall	8.3 ± 1.2	21.6 ± 6.7	44.4 ± 12.3	73.3 ± 9.8	88.6 ± 5.7
$\varepsilon=8$	precision	58.4 ± 4.6	56.4 ± 2.0	52.3 ± 3.2	38.4 ± 5.3	22.7 ± 5.3
	recall	16.2 ± 3.9	31.1 ± 7.4	56.5 ± 10.0	81.6 ± 9.1	94.3 ± 2.8
$\varepsilon=9$	precision	77.9 ± 2.7	71.3 ± 2.9	61.3 ± 3.6	41.8 ± 7.7	23.1 ± 5.6
	recall	21.5 ± 4.3	39.0 ± 8.6	66.4 ± 12.0	87.9 ± 6.9	95.6 ± 2.1
$\varepsilon=10$	precision	79.5 ± 1.4	77.9 ± 3.4	64.7 ± 5.8	42.7 ± 9.4	23.1 ± 5.9
	recall	21.9 ± 4.5	42.4 ± 7.9	69.5 ± 10.9	89.0 ± 5.2	95.4 ± 3.0

high		$\hat{k}=k/4$	$\hat{k}=k/2$	$\hat{k}=k$	$\hat{k}=2k$	$\hat{k}=4k$
$\varepsilon=1$	precision	4.2 ± 0.9	4.1 ± 1.2	4.0 ± 1.5	3.0 ± 0.8	2.9 ± 0.5
	recall	1.1 ± 0.2	2.1 ± 0.6	4.1 ± 1.5	6.0 ± 1.7	11.9 ± 2.1
$\varepsilon=2$	precision	6.1 ± 2.1	7.2 ± 2.9	7.0 ± 1.6	6.7 ± 0.7	6.0 ± 0.6
	recall	1.5 ± 0.5	3.7 ± 1.5	7.1 ± 1.6	13.7 ± 1.4	24.4 ± 2.2
$\varepsilon=3$	precision	19.9 ± 5.3	18.6 ± 1.5	17.0 ± 0.9	16.8 ± 0.6	15.2 ± 0.6
	recall	5.1 ± 1.3	9.5 ± 0.7	17.3 ± 0.9	34.3 ± 1.0	61.9 ± 2.2
$\varepsilon=4$	precision	29.7 ± 3.7	29.5 ± 2.2	31.2 ± 1.7	31.8 ± 1.6	21.4 ± 0.3
	recall	7.6 ± 0.9	15.0 ± 1.0	31.8 ± 1.6	64.9 ± 3.2	87.3 ± 1.6
$\varepsilon=5$	precision	48.0 ± 2.0	51.5 ± 1.7	55.7 ± 1.4	44.6 ± 0.6	23.6 ± 0.2
	recall	12.2 ± 0.6	26.2 ± 1.0	56.7 ± 1.3	90.8 ± 1.2	96.2 ± 0.5
$\varepsilon=6$	precision	71.9 ± 3.1	75.6 ± 3.3	76.9 ± 2.3	46.5 ± 0.5	24.1 ± 0.2
	recall	18.3 ± 0.9	38.5 ± 1.8	78.4 ± 2.4	94.7 ± 1.0	98.1 ± 0.7
$\varepsilon=7$	precision	88.2 ± 3.3	89.8 ± 1.3	81.5 ± 1.6	47.2 ± 0.6	24.3 ± 0.1
	recall	22.5 ± 0.8	45.7 ± 0.7	83.1 ± 1.4	96.0 ± 0.9	98.9 ± 0.6
$\varepsilon=8$	precision	94.4 ± 1.6	93.5 ± 1.0	80.0 ± 2.3	47.0 ± 0.7	24.3 ± 0.1
	recall	24.0 ± 0.5	47.6 ± 0.7	81.5 ± 2.2	95.8 ± 1.1	98.8 ± 0.2
$\varepsilon=9$	precision	97.4 ± 0.2	95.1 ± 0.8	81.2 ± 2.2	47.5 ± 0.8	24.3 ± 0.2
	recall	24.8 ± 0.2	48.4 ± 0.5	82.7 ± 2.2	96.7 ± 1.4	98.8 ± 0.4
$\varepsilon=10$	precision	98.4 ± 1.0	96.1 ± 1.0	81.7 ± 2.0	47.5 ± 0.5	24.3 ± 0.1
	recall	25.1 ± 0.3	49.0 ± 0.6	83.2 ± 2.1	96.7 ± 0.7	99.0 ± 0.2

TABLE XX: twitch-ENGB (LAPGRAPH)

low		$\hat{k}=k/4$	$\hat{k}=k/2$	$\hat{k}=k$	$\hat{k}=2k$	$\hat{k}=4k$
$\varepsilon=1$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon=2$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon=3$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon=4$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon=5$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.7 ± 0.9	0.3 ± 0.5
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1.7 ± 2.4	1.7 ± 2.4
$\varepsilon=6$	precision	8.3 ± 11.8	4.8 ± 6.7	5.2 ± 3.7	6.0 ± 1.6	3.7 ± 0.5
	recall	2.1 ± 2.9	2.1 ± 2.9	5.4 ± 4.1	12.2 ± 4.2	14.3 ± 1.3
$\varepsilon=7$	precision	13.1 ± 10.2	12.1 ± 12.1	7.8 ± 6.3	8.7 ± 2.5	9.0 ± 2.2
	recall	3.8 ± 2.7	5.8 ± 5.1	7.5 ± 5.4	16.4 ± 1.7	34.4 ± 2.8
$\varepsilon=8$	precision	38.1 ± 16.8	36.3 ± 9.3	22.1 ± 1.5	21.3 ± 1.9	15.3 ± 3.4
	recall	11.0 ± 4.4	18.9 ± 3.3	22.3 ± 2.6	42.5 ± 8.1	58.8 ± 3.6
$\varepsilon=9$	precision	51.2 ± 19.0	46.2 ± 8.0	35.2 ± 6.2	28.0 ± 0.0	17.3 ± 3.4
	recall	15.2 ± 5.7	24.8 ± 4.9	36.9 ± 13.0	55.9 ± 10.8	66.8 ± 3.7
$\varepsilon=10$	precision	47.6 ± 3.4	41.8 ± 10.3	40.4 ± 8.8	35.3 ± 6.6	21.0 ± 3.7
	recall	14.3 ± 1.3	23.0 ± 8.6	42.3 ± 16.0	69.2 ± 12.7	81.1 ± 3.3

unconstrained		$\hat{k}=k/4$	$\hat{k}=k/2$	$\hat{k}=k$	$\hat{k}=2k$	$\hat{k}=4k$
$\varepsilon=1$	precision	0.5 ± 0.7	0.3 ± 0.4	0.1 ± 0.2	0.2 ± 0.2	0.2 ± 0.2
	recall	0.2 ± 0.2	0.2 ± 0.2	0.2 ± 0.2	0.5 ± 0.4	1.0 ± 1.0
$\varepsilon=2$	precision	1.1 ± 1.5	0.5 ± 0.8	0.4 ± 0.6	0.3 ± 0.4	0.3 ± 0.1
	recall	0.3 ± 0.5	0.3 ± 0.5	0.5 ± 0.7	0.7 ± 0.9	1.2 ± 0.6
$\varepsilon=3$	precision	3.7 ± 2.7	2.7 ± 0.7	2.1 ± 0.8	1.3 ± 0.6	0.8 ± 0.3
	recall	0.9 ± 0.6	1.4 ± 0.4	2.1 ± 0.4	2.5 ± 0.7	3.4 ± 0.9
$\varepsilon=4$	precision	5.3 ± 1.5	6.1 ± 2.5	4.9 ± 1.5	3.0 ± 1.2	1.8 ± 0.6
	recall	1.4 ± 0.4	3.4 ± 1.7	5.2 ± 1.8	6.0 ± 1.7	7.4 ± 1.8
$\varepsilon=5$	precision	11.6 ± 5.9	12.5 ± 5.6	12.0 ± 2.7	7.4 ± 1.6	4.3 ± 1.2
	recall	3.0 ± 1.6	6.9 ± 3.8	12.9 ± 3.7	15.8 ± 3.7	18.1 ± 4.2
$\varepsilon=6$	precision	21.0 ± 3.9	24.7 ± 5.2	21.5 ± 1.5	14.8 ± 1.4	8.9 ± 1.5
	recall	5.8 ± 1.5	13.7 ± 4.5	23.7 ± 6.2	31.7 ± 5.0	37.5 ± 3.9
$\varepsilon=7$	precision	34.8 ± 6.1	34.3 ± 3.7	33.5 ± 0.5	23.2 ± 3.4	13.8 ± 3.0
	recall	9.9 ± 3.3	19.2 ± 5.5	36.7 ± 8.3	49.0 ± 5.0	57.5 ± 2.6
$\varepsilon=8$	precision	46.3 ± 1.8	49.2 ± 2.3	45.7 ± 2.8	31.0 ± 4.9	17.6 ± 4.4
	recall	13.0 ± 3.2	27.0 ± 5.9	49.5 ± 9.1	65.4 ± 6.3	72.8 ± 1.0
$\varepsilon=9$	precision	64.2 ± 6.7	64.4 ± 2.7	52.7 ± 1.8	35.6 ± 6.8	19.9 ± 4.8
	recall	17.4 ± 2.6	35.2 ± 7.5	57.3 ± 11.3	74.5 ± 4.6	82.2 ± 1.7
$\varepsilon=10$	precision	75.8 ± 4.2	71.8 ± 3.4	61.2 ± 4.9	40.0 ± 7.9	21.9 ± 5.7
	recall	20.9 ± 4.1	39.3 ± 8.2	65.9 ± 10.5	83.8 ± 4.8	90.3 ± 0.4

high		$\hat{k}=k/4$	$\hat{k}=k/2$	$\hat{k}=k$	$\hat{k}=2k$	$\hat{k}=4k$
$\varepsilon=1$	precision	1.7 ± 0.4	1.2 ± 0.3	1.1 ± 0.3	0.9 ± 0.2	0.4 ± 0.1
	recall	0.4 ± 0.1	0.6 ± 0.1	1.1 ± 0.4	1.7 ± 0.3	1.7 ± 0.3
$\varepsilon=2$	precision	3.2 ± 0.5	2.0 ± 0.4	1.5 ± 0.2	1.1 ± 0.2	0.6 ± 0.1
	recall	0.8 ± 0.1	1.0 ± 0.2	1.6 ± 0.2	2.3 ± 0.3	2.3 ± 0.3
$\varepsilon=3$	precision	9.3 ± 2.1	5.4 ± 1.2	3.4 ± 0.8	2.2 ± 0.4	1.1 ± 0.2
	recall	2.4 ± 0.5	2.8 ± 0.6	3.5 ± 0.8	4.5 ± 0.7	4.5 ± 0.7
$\varepsilon=4$	precision	24.7 ± 2.5	13.2 ± 1.4	7.9 ± 0.8	5.1 ± 0.5	3.1 ± 0.9
	recall	6.3 ± 0.6	6.7 ± 0.7	8.0 ± 0.7	10.5 ± 0.9	12.5 ± 3.6
$\varepsilon=5$	precision	50.1 ± 1.4	30.9 ± 1.3	17.4 ± 1.0	10.5 ± 1.0	6.3 ± 0.3
	recall	12.8 ± 0.3	15.8 ± 0.6	17.7 ± 0.9	21.4 ± 1.9	25.7 ± 1.3
$\varepsilon=6$	precision	76.0 ± 1.2	61.0 ± 0.6	35.2 ± 0.5	19.9 ± 0.5	11.9 ± 0.4
	recall	19.4 ± 0.2	31.1 ± 0.4	35.8 ± 0.4	40.5 ± 0.8	48.3 ± 1.6
$\varepsilon=7$	precision	88.3 ± 1.5	83.3 ± 1.9	56.3 ± 1.2	31.4 ± 0.9	17.3 ± 0.5
	recall	22.5 ± 0.3	42.5 ± 1.0	57.4 ± 1.0	64.0 ± 1.5	70.3 ± 1.7
$\varepsilon=8$	precision	93.0 ± 0.7	89.7 ± 1.9	67.3 ± 1.7	37.9 ± 0.8	20.2 ± 0.5
	recall	23.7 ± 0.1	45.7 ± 1.0	68.6 ± 1.6	77.2 ± 1.3	82.1 ± 1.6
$\varepsilon=9$	precision	96.0 ± 1.4	92.5 ± 0.5	73.3 ± 1.7	41.9 ± 0.8	21.9 ± 0.3
	recall	24.4 ± 0.4	47.1 ± 0.3	74.7 ± 1.6	85.3 ± 1.4	89.0 ± 1.0
$\varepsilon=10$	precision	97.3 ± 0.6	94.2 ± 1.0	77.3 ± 2.2	44.3 ± 0.4	22.9 ± 0.2
	recall	24.8 ± 0.2	48.0 ± 0.4	78.8 ± 2.1	90.3 ± 0.6	93.3 ± 0.6

TABLE XXI: twitch-PTBR (EDGERAND)

low		$\hat{k} = k/4$	$\hat{k} = k/2$	$\hat{k} = k$	$\hat{k} = 2k$	$\hat{k} = 4k$
$\varepsilon = 1$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon = 2$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.3 ± 0.5	0.3 ± 0.5
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.6 ± 0.9	1.3 ± 1.8
$\varepsilon = 3$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.3 ± 0.5	0.2 ± 0.2
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.6 ± 0.9	0.6 ± 0.9
$\varepsilon = 4$	precision	0.0 ± 0.0	1.3 ± 1.9	0.7 ± 0.9	1.2 ± 0.9	2.0 ± 0.6
	recall	0.0 ± 0.0	0.6 ± 0.9	0.6 ± 0.9	2.4 ± 1.7	8.3 ± 2.2
$\varepsilon = 5$	precision	0.0 ± 0.0	2.1 ± 1.5	4.3 ± 2.7	4.1 ± 1.5	5.1 ± 0.1
	recall	0.0 ± 0.0	1.1 ± 0.8	4.3 ± 2.5	8.4 ± 2.6	21.4 ± 1.5
$\varepsilon = 6$	precision	13.9 ± 6.9	13.7 ± 3.0	13.0 ± 1.0	13.8 ± 2.1	14.2 ± 1.2
	recall	3.6 ± 1.6	7.2 ± 1.5	13.6 ± 1.7	28.9 ± 5.5	59.1 ± 7.8
$\varepsilon = 7$	precision	40.4 ± 6.5	31.2 ± 2.6	32.2 ± 1.6	28.9 ± 2.2	23.1 ± 1.3
	recall	10.8 ± 1.6	16.5 ± 2.2	33.8 ± 3.5	60.3 ± 7.3	95.9 ± 0.7
$\varepsilon = 8$	precision	60.1 ± 2.7	53.5 ± 3.9	59.7 ± 0.7	45.3 ± 1.3	24.1 ± 1.4
	recall	16.1 ± 1.0	28.4 ± 3.8	62.3 ± 3.3	93.9 ± 2.6	100.0± 0.0
$\varepsilon = 9$	precision	77.7 ± 2.6	80.8 ± 0.6	69.8 ± 2.3	47.4 ± 2.6	23.9 ± 1.5
	recall	20.8 ± 1.2	42.7 ± 3.1	73.0 ± 6.3	98.2 ± 0.1	98.9 ± 0.8
$\varepsilon = 10$	precision	90.7 ± 4.3	91.1 ± 2.6	79.6 ± 1.3	47.1 ± 2.2	24.1 ± 1.4
	recall	24.3 ± 2.3	48.1 ± 3.5	83.1 ± 5.1	97.6 ± 1.0	100.0± 0.0

unconstrained		$\hat{k} = k/4$	$\hat{k} = k/2$	$\hat{k} = k$	$\hat{k} = 2k$	$\hat{k} = 4k$
$\varepsilon = 1$	precision	5.2 ± 2.3	3.4 ± 1.0	3.4 ± 1.2	3.6 ± 1.0	4.0 ± 0.8
	recall	1.4 ± 0.8	1.8 ± 0.8	3.8 ± 1.9	8.0 ± 3.5	17.5 ± 6.9
$\varepsilon = 2$	precision	7.3 ± 1.6	6.9 ± 2.1	7.3 ± 1.5	8.9 ± 0.1	10.1 ± 0.5
	recall	2.0 ± 0.7	3.8 ± 1.7	8.0 ± 3.2	18.6 ± 4.6	42.0 ± 9.7
$\varepsilon = 3$	precision	11.5 ± 2.1	15.7 ± 1.2	18.3 ± 0.2	20.4 ± 1.0	17.7 ± 1.3
	recall	3.1 ± 1.2	8.3 ± 2.4	19.2 ± 4.8	42.9 ± 11.2	73.3 ± 15.9
$\varepsilon = 4$	precision	24.5 ± 4.4	28.8 ± 1.4	34.5 ± 3.6	33.0 ± 2.1	22.6 ± 4.8
	recall	6.7 ± 2.6	15.3 ± 4.3	37.0 ± 11.9	70.0 ± 19.9	89.6 ± 6.9
$\varepsilon = 5$	precision	41.3 ± 6.0	49.4 ± 4.9	51.9 ± 2.8	40.5 ± 5.7	24.0 ± 6.1
	recall	11.2 ± 3.9	26.5 ± 8.4	54.9 ± 15.5	81.9 ± 11.2	94.1 ± 3.9
$\varepsilon = 6$	precision	66.0 ± 5.5	70.8 ± 3.7	65.7 ± 2.8	45.5 ± 9.6	25.1 ± 7.0
	recall	17.7 ± 5.5	37.6 ± 10.8	68.2 ± 15.0	90.4 ± 7.0	97.8 ± 1.7
$\varepsilon = 7$	precision	84.4 ± 2.3	83.7 ± 2.0	73.6 ± 6.3	47.7 ± 11.9	25.4 ± 7.4
	recall	22.2 ± 5.8	44.0 ± 11.3	75.6 ± 14.0	93.8 ± 4.2	98.7 ± 0.8
$\varepsilon = 8$	precision	92.0 ± 1.3	90.9 ± 1.7	77.1 ± 8.4	48.2 ± 11.8	25.3 ± 7.3
	recall	24.1 ± 5.9	47.5 ± 11.5	78.6 ± 12.9	94.8 ± 4.6	98.7 ± 1.1
$\varepsilon = 9$	precision	95.4 ± 1.4	93.2 ± 1.1	78.4 ± 8.1	48.7 ± 12.2	25.4 ± 7.4
	recall	25.1 ± 6.5	48.9 ± 12.2	80.1 ± 13.6	95.6 ± 4.1	98.9 ± 0.9
$\varepsilon = 10$	precision	95.6 ± 0.8	93.7 ± 1.3	78.3 ± 9.0	48.7 ± 12.4	25.4 ± 7.4
	recall	25.1 ± 6.4	49.1 ± 12.1	79.8 ± 12.8	95.5 ± 3.8	99.0 ± 0.9

high		$\hat{k} = k/4$	$\hat{k} = k/2$	$\hat{k} = k$	$\hat{k} = 2k$	$\hat{k} = 4k$
$\varepsilon = 1$	precision	5.2 ± 2.3	3.4 ± 1.0	3.4 ± 1.2	3.6 ± 1.0	4.0 ± 0.8
	recall	1.4 ± 0.8	1.8 ± 0.8	3.8 ± 1.9	8.0 ± 3.5	17.5 ± 6.9
$\varepsilon = 2$	precision	7.3 ± 1.6	6.9 ± 2.1	7.3 ± 1.5	8.9 ± 0.1	10.1 ± 0.5
	recall	2.0 ± 0.7	3.8 ± 1.7	8.0 ± 3.2	18.6 ± 4.6	42.0 ± 9.7
$\varepsilon = 3$	precision	11.5 ± 2.1	15.7 ± 1.2	18.3 ± 0.2	20.4 ± 1.0	17.7 ± 1.3
	recall	3.1 ± 1.2	8.3 ± 2.4	19.2 ± 4.8	42.9 ± 11.2	73.3 ± 15.9
$\varepsilon = 4$	precision	24.5 ± 4.4	28.8 ± 1.4	34.5 ± 3.6	33.0 ± 2.1	22.6 ± 4.8
	recall	6.7 ± 2.6	15.3 ± 4.3	37.0 ± 11.9	70.0 ± 19.9	89.6 ± 6.9
$\varepsilon = 5$	precision	41.3 ± 6.0	49.4 ± 4.9	51.9 ± 2.8	40.5 ± 5.7	24.0 ± 6.1
	recall	11.2 ± 3.9	26.5 ± 8.4	54.9 ± 15.5	81.9 ± 11.2	94.1 ± 3.9
$\varepsilon = 6$	precision	66.0 ± 5.5	70.8 ± 3.7	65.7 ± 2.8	45.5 ± 9.6	25.1 ± 7.0
	recall	17.7 ± 5.5	37.6 ± 10.8	68.2 ± 15.0	90.4 ± 7.0	97.8 ± 1.7
$\varepsilon = 7$	precision	84.4 ± 2.3	83.7 ± 2.0	73.6 ± 6.3	47.7 ± 11.9	25.4 ± 7.4
	recall	22.2 ± 5.8	44.0 ± 11.3	75.6 ± 14.0	93.8 ± 4.2	98.7 ± 0.8
$\varepsilon = 8$	precision	92.0 ± 1.3	90.9 ± 1.7	77.1 ± 8.4	48.2 ± 11.8	25.3 ± 7.3
	recall	24.1 ± 5.9	47.5 ± 11.5	78.6 ± 12.9	94.8 ± 4.6	98.7 ± 1.1
$\varepsilon = 9$	precision	95.4 ± 1.4	93.2 ± 1.1	78.4 ± 8.1	48.7 ± 12.2	25.4 ± 7.4
	recall	25.1 ± 6.5	48.9 ± 12.2	80.1 ± 13.6	95.6 ± 4.1	98.9 ± 0.9
$\varepsilon = 10$	precision	95.6 ± 0.8	93.7 ± 1.3	78.3 ± 9.0	48.7 ± 12.4	25.4 ± 7.4
	recall	25.1 ± 6.4	49.1 ± 12.1	79.8 ± 12.8	95.5 ± 3.8	99.0 ± 0.9

TABLE XXII: twitch-PTBR (LAPGRAPH)

low		$\hat{k} = k/4$	$\hat{k} = k/2$	$\hat{k} = k$	$\hat{k} = 2k$	$\hat{k} = 4k$
$\varepsilon = 1$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.3 ± 0.2
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1.1 ± 0.8
$\varepsilon = 2$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.5 ± 0.4	0.5 ± 0.4
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1.1 ± 0.8	2.3 ± 1.6
$\varepsilon = 3$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.3 ± 0.2
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1.1 ± 0.8
$\varepsilon = 4$	precision	2.1 ± 2.9	1.0 ± 1.5	2.3 ± 0.7	2.7 ± 1.0	1.8 ± 0.5
	recall	0.6 ± 0.8	0.6 ± 0.8	2.4 ± 0.8	5.4 ± 1.8	7.2 ± 1.8
$\varepsilon = 5$	precision	4.6 ± 3.3	3.4 ± 2.6	2.4 ± 1.7	4.0 ± 0.7	4.3 ± 0.5
	recall	1.2 ± 0.9	1.8 ± 1.4	2.5 ± 1.7	8.3 ± 1.6	17.9 ± 2.4
$\varepsilon = 6$	precision	6.7 ± 0.7	6.8 ± 0.8	10.2 ± 0.7	10.2 ± 0.9	8.9 ± 0.1
	recall	1.8 ± 0.1	3.6 ± 0.2	10.7 ± 0.9	21.3 ± 3.0	36.8 ± 1.7
$\varepsilon = 7$	precision	22.8 ± 5.7	17.1 ± 2.1	15.0 ± 2.2	14.4 ± 1.7	13.9 ± 0.9
	recall	6.0 ± 1.2	9.0 ± 0.5	15.5 ± 1.4	30.1 ± 5.1	58.1 ± 6.8
$\varepsilon = 8$	precision	27.4 ± 8.2	24.2 ± 5.7	23.3 ± 2.0	22.5 ± 3.9	19.7 ± 1.6
	recall	7.2 ± 1.9	12.6 ± 2.1	24.4 ± 2.6	47.1 ± 10.2	82.2 ± 10.5
$\varepsilon = 9$	precision	22.3 ± 2.6	27.0 ± 1.4	27.2 ± 2.1	30.1 ± 1.5	21.9 ± 0.2
	recall	5.9 ± 0.6	14.3 ± 0.7	28.4 ± 3.0	62.8 ± 6.4	90.9 ± 4.5
$\varepsilon = 10$	precision	40.9 ± 9.5	39.7 ± 3.1	39.0 ± 2.9	39.3 ± 1.9	23.1 ± 0.7
	recall	10.8 ± 2.0	20.8 ± 0.4	40.8 ± 4.6	81.8 ± 7.5	95.7 ± 2.5

unconstrained		$\hat{k} = k/4$	$\hat{k} = k/2$	$\hat{k} = k$	$\hat{k} = 2k$	$\hat{k} = 4k$
$\varepsilon = 1$	precision	6.1 ± 0.8	5.0 ± 0.3	4.2 ± 0.4	3.6 ± 0.6	2.9 ± 0.3
	recall	1.6 ± 0.3	2.6 ± 0.6	4.4 ± 1.0	7.4 ± 1.1	12.1 ± 2.7
$\varepsilon = 2$	precision	12.0 ± 1.8	11.0 ± 0.9	9.6 ± 0.7	7.2 ± 1.2	4.8 ± 0.6
	recall	3.1 ± 0.6	5.6 ± 1.1	10.1 ± 2.6	14.4 ± 1.9	19.6 ± 3.2
$\varepsilon = 3$	precision	18.6 ± 1.3	19.8 ± 1.1	19.8 ± 1.2	14.1 ± 3.3	8.5 ± 1.6
	recall	4.8 ± 1.1	10.5 ± 2.9	20.8 ± 5.2	27.9 ± 1.9	33.8 ± 3.5
$\varepsilon = 4$	precision	25.0 ± 1.8	33.2 ± 3.8	33.8 ± 2.9	23.9 ± 5.5	13.7 ± 3.3
	recall	6.7 ± 2.0	17.9 ± 5.9	35.6 ± 9.7	47.3 ± 3.0	54.1 ± 3.0
$\varepsilon = 5$	precision	41.3 ± 0.9	49.7 ± 4.2	47.9 ± 2.8	32.8 ± 6.8	18.5 ± 4.5
	recall	10.9 ± 2.9	26.5 ± 8.1	49.7 ± 10.8	65.3 ± 5.2	73.0 ± 3.8
$\varepsilon = 6$	precision	55.2 ± 6.3	62.6 ± 4.0	58.6 ± 4.0	39.0 ± 9.0	21.7 ± 5.7
	recall	14.9 ± 4.9	33.3 ± 9.7	60.5 ± 12.3	77.0 ± 4.7	84.9 ± 2.8
$\varepsilon = 7$	precision	66.6 ± 6.5	71.8 ± 5.1	66.0 ± 5.1	42.8 ± 10.3	23.1 ± 6.2
	recall	17.8 ± 5.6	38.2 ± 11.1	67.9 ± 13.1	84.4 ± 4.4	90.4 ± 2.4
$\varepsilon = 8$	precision	75.1 ± 4.9	78.0 ± 3.4	68.6 ± 5.1	44.6 ± 10.5	24.0 ± 6.6
	recall	20.0 ± 5.8	41.2 ± 11.0	70.7 ± 14.0	88.1 ± 5.1	93.9 ± 2.2
$\varepsilon = 9$	precision	81.5 ± 5.5	84.1 ± 3.3	73.2 ± 5.7	46.0 ± 10.4	24.7 ± 6.9
	recall	21.7 ± 6.5	44.5 ± 12.2	75.3 ± 14.3	90.9 ± 5.8	96.2 ± 1.6
$\varepsilon = 10$	precision	87.2 ± 2.2	88.2 ± 1.5	75.8 ± 7.4	47.1 ± 11.5	24.9 ± 7.0
	recall	23.0 ± 6.1	46.4 ± 12.0	77.6 ± 13.6	92.7 ± 4.6	96.9 ± 1.6

high		$\hat{k} = k/4$	$\hat{k} = k/2$	$\hat{k} = k$	$\hat{k} = 2k$	$\hat{k} = 4k$
$\varepsilon = 1$	precision	10.3 ± 0.9	9.1 ± 0.1	8.3 ± 0.5	6.8 ± 0.4	5.4 ± 0.5
	recall	2.5 ± 0.2	4.5 ± 0.6	8.1 ± 0.7	13.3 ± 1.3	21.0 ± 2.2
$\varepsilon = 2$	precision	21.8 ± 1.4	19.9 ± 0.8	14.7 ± 0.7	10.4 ± 0.5	7.6 ± 0.6
	recall	5.4 ± 0.4	9.8 ± 0.9	14.5 ± 1.0	20.4 ± 1.7	29.7 ± 3.3
$\varepsilon = 3$	precision	40.2 ± 2.8	37.8 ± 2.2	28.7 ± 2.1	18.0 ± 1.5	11.3 ± 1.0
	recall	9.8 ± 0.6	18.5 ± 1.5	28.0 ± 1.3	35.2 ± 1.2	44.3 ± 1.9
$\varepsilon = 4$	precision	53.9 ± 1.3	58.4 ± 1.5	48.4 ± 4.9	28.2 ± 2.8	16.3 ± 1.5
	recall	13.2 ± 1.2	28.7 ± 2.8	47.2 ± 1.0	54.9 ± 1.3	63.7 ± 1.6
$\varepsilon = 5$	precision	70.0 ± 0.6	73.2 ± 1.4	64.0 ± 5.2	37.3 ± 3.7	20.5 ± 2.0
	recall	17.2 ± 1.8	36.0 ± 3.6	62.5 ± 2.1	72.7 ± 1.3	79.9 ± 1.6
$\varepsilon = 6$	precision	79.9 ± 1.1	82.0 ± 1.7	74.2 ± 5.7	42.9 ± 4.3	22.9 ± 2.4
	recall	19.7 ± 2.2	40.4 ± 3.9	72.5 ± 2.7	83.6 ± 1.0	89.2 ± 0.9
$\varepsilon = 7$	precision	86.8 ± 0.7	89.1 ± 0.7	80.0 ± 5.9	46.0 ± 5.0	24.0 ± 2.6
	recall	21.4 ± 2.3	43.9 ± 4.8	78.2 ± 3.3	89.6 ± 0.3	93.7 ± 0.3
$\varepsilon = 8$	precision	89.8 ± 1.5	90.6 ± 2.1	79.5 ± 5.4	47.5 ± 5.0	24.6 ± 2.7
	recall	22.1 ± 2.3	44.6 ± 4.1	77.8 ± 3.6	92.5 ± 0.6	95.9 ± 0.4
$\varepsilon = 9$	precision	93.5 ± 1.3	94.4 ± 1.2	83.5 ± 5.9	48.5 ± 5.1	25.0 ± 2.7
	recall	23.0 ± 2.3	46.5 ± 4.7	81.6 ± 3.7	94.6 ± 0.7	97.3 ± 0.2
$\varepsilon = 10$	precision	95.4 ± 0.9	95.8 ± 1.0	85.5 ± 6.3	49.5 ± 5.3	25.2 ± 2.8
	recall	23.5 ± 2.5	47.2 ± 4.9	83.6 ± 3.7	96.4 ± 0.4	98.3 ± 0.1

TABLE XXIII: PPI (EDGERAND)

low		$\hat{k}=k/4$	$\hat{k}=k/2$	$\hat{k}=k$	$\hat{k}=2k$	$\hat{k}=4k$
$\varepsilon=1$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon=2$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon=3$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon=4$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon=5$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1.0 ± 0.8
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	3.7 ± 2.8
$\varepsilon=6$	precision	0.0 ± 0.0	0.0 ± 0.0	2.7 ± 3.8	3.3 ± 2.5	4.3 ± 3.1
	recall	0.0 ± 0.0	0.0 ± 0.0	2.8 ± 3.9	6.2 ± 4.5	16.4 ± 11.6
$\varepsilon=7$	precision	4.8 ± 6.7	5.1 ± 3.6	7.9 ± 3.3	14.7 ± 7.5	15.3 ± 6.6
	recall	1.4 ± 2.0	2.5 ± 1.8	7.4 ± 3.8	27.5 ± 15.3	57.0 ± 27.7
$\varepsilon=8$	precision	14.3 ± 11.7	24.9 ± 6.8	26.4 ± 4.0	32.7 ± 7.4	27.7 ± 2.6
	recall	3.9 ± 3.4	12.0 ± 4.0	24.3 ± 6.5	60.1 ± 20.7	97.9 ± 2.9
$\varepsilon=9$	precision	51.2 ± 19.4	55.3 ± 15.3	54.1 ± 7.4	52.7 ± 4.1	28.3 ± 3.3
	recall	13.4 ± 5.1	26.1 ± 6.5	48.9 ± 8.1	93.8 ± 8.8	100.0± 0.0
$\varepsilon=10$	precision	59.5 ± 8.9	77.7 ± 5.4	78.9 ± 4.1	56.7 ± 6.6	28.3 ± 3.3
	recall	15.7 ± 3.7	37.2 ± 6.3	71.1 ± 5.8	100.0± 0.0	100.0± 0.0

unconstrained		$\hat{k}=k/4$	$\hat{k}=k/2$	$\hat{k}=k$	$\hat{k}=2k$	$\hat{k}=4k$
$\varepsilon=1$	precision	0.0 ± 0.0	0.5 ± 0.8	0.4 ± 0.3	0.5 ± 0.2	0.6 ± 0.2
	recall	0.0 ± 0.0	0.2 ± 0.3	0.4 ± 0.3	1.1 ± 0.5	2.5 ± 0.6
$\varepsilon=2$	precision	1.6 ± 2.2	1.6 ± 1.1	1.6 ± 0.7	1.6 ± 0.7	1.1 ± 0.4
	recall	0.5 ± 0.7	0.9 ± 0.7	1.7 ± 0.8	3.4 ± 2.0	4.7 ± 1.7
$\varepsilon=3$	precision	3.7 ± 2.0	4.5 ± 1.6	4.0 ± 1.6	3.3 ± 1.1	3.2 ± 0.7
	recall	1.0 ± 0.6	2.4 ± 1.1	4.3 ± 2.1	6.7 ± 2.2	12.8 ± 1.7
$\varepsilon=4$	precision	8.5 ± 2.0	6.4 ± 2.6	6.9 ± 1.5	7.4 ± 1.3	6.6 ± 1.5
	recall	2.3 ± 0.8	3.4 ± 1.7	7.2 ± 1.7	15.1 ± 1.8	26.7 ± 3.3
$\varepsilon=5$	precision	9.5 ± 3.4	12.3 ± 2.3	12.5 ± 3.0	12.5 ± 2.1	19.1 ± 1.1
	recall	2.6 ± 1.2	6.4 ± 1.8	12.8 ± 2.4	25.5 ± 3.0	78.4 ± 4.5
$\varepsilon=6$	precision	20.1 ± 5.8	20.8 ± 6.0	24.9 ± 1.6	41.5 ± 2.8	24.4 ± 2.7
	recall	5.2 ± 1.4	10.6 ± 2.4	25.8 ± 2.9	85.7 ± 9.9	99.7 ± 0.2
$\varepsilon=7$	precision	33.3 ± 5.6	42.9 ± 5.1	59.7 ± 3.9	48.8 ± 5.5	24.4 ± 2.7
	recall	8.5 ± 0.7	21.9 ± 0.7	61.5 ± 3.8	99.7 ± 0.2	99.7 ± 0.2
$\varepsilon=8$	precision	54.5 ± 6.5	70.1 ± 4.0	78.0 ± 2.8	48.8 ± 5.5	24.4 ± 2.7
	recall	14.3 ± 3.0	36.4 ± 5.4	80.5 ± 6.9	99.7 ± 0.2	99.7 ± 0.2
$\varepsilon=9$	precision	82.5 ± 3.4	88.3 ± 0.4	85.7 ± 4.4	49.0 ± 5.4	24.5 ± 2.7
	recall	21.6 ± 3.2	45.7 ± 5.1	88.3 ± 5.4	100.0± 0.0	100.0± 0.0
$\varepsilon=10$	precision	90.5 ± 2.2	93.3 ± 1.0	87.9 ± 5.6	49.0 ± 5.4	24.5 ± 2.7
	recall	23.7 ± 3.2	48.4 ± 5.7	90.4 ± 4.3	100.0± 0.0	100.0± 0.0

high		$\hat{k}=k/4$	$\hat{k}=k/2$	$\hat{k}=k$	$\hat{k}=2k$	$\hat{k}=4k$
$\varepsilon=1$	precision	2.5 ± 1.1	2.2 ± 0.6	2.6 ± 0.2	3.0 ± 0.2	3.2 ± 0.1
	recall	0.5 ± 0.2	0.9 ± 0.3	2.3 ± 0.1	5.2 ± 0.2	11.1 ± 0.7
$\varepsilon=2$	precision	8.0 ± 2.0	8.5 ± 1.7	8.1 ± 1.0	8.1 ± 0.6	7.9 ± 0.4
	recall	1.7 ± 0.4	3.7 ± 0.6	7.0 ± 0.6	14.0 ± 0.5	27.4 ± 0.2
$\varepsilon=3$	precision	19.2 ± 3.3	19.6 ± 0.9	18.9 ± 1.1	18.0 ± 1.2	18.0 ± 0.6
	recall	4.2 ± 0.6	8.5 ± 0.2	16.3 ± 0.6	31.1 ± 1.0	62.3 ± 0.5
$\varepsilon=4$	precision	38.9 ± 1.7	38.8 ± 1.6	36.6 ± 1.4	35.5 ± 0.8	28.3 ± 1.1
	recall	8.4 ± 0.3	16.8 ± 0.4	31.8 ± 0.6	61.6 ± 1.5	98.3 ± 0.1
$\varepsilon=5$	precision	59.8 ± 1.8	58.1 ± 0.9	58.6 ± 0.4	57.3 ± 2.3	28.7 ± 1.2
	recall	13.0 ± 0.5	25.2 ± 0.7	50.9 ± 1.7	99.4 ± 0.1	99.6 ± 0.1
$\varepsilon=6$	precision	77.1 ± 2.1	77.3 ± 1.5	79.4 ± 1.3	57.6 ± 2.4	28.8 ± 1.2
	recall	16.7 ± 0.7	33.6 ± 1.4	69.0 ± 2.2	99.9 ± 0.1	99.9 ± 0.1
$\varepsilon=7$	precision	89.3 ± 2.4	90.0 ± 1.2	91.6 ± 1.2	57.7 ± 2.3	28.8 ± 1.2
	recall	19.4 ± 0.7	39.1 ± 1.4	79.6 ± 3.1	100.0± 0.0	100.0± 0.0
$\varepsilon=8$	precision	95.4 ± 1.3	96.0 ± 1.0	95.7 ± 0.7	57.7 ± 2.3	28.8 ± 1.2
	recall	20.7 ± 0.7	41.7 ± 1.4	83.1 ± 3.1	100.0± 0.0	100.0± 0.0
$\varepsilon=9$	precision	97.2 ± 0.4	97.9 ± 0.3	97.0 ± 0.2	57.7 ± 2.3	28.8 ± 1.2
	recall	21.1 ± 0.9	42.5 ± 1.8	84.3 ± 3.4	100.0± 0.0	100.0± 0.0
$\varepsilon=10$	precision	99.0 ± 0.5	99.1 ± 0.1	97.6 ± 0.2	57.7 ± 2.3	28.8 ± 1.2
	recall	21.5 ± 0.8	43.1 ± 1.7	84.8 ± 3.2	100.0± 0.0	100.0± 0.0

TABLE XXIV: PPI (LAPGRAPH)

low		$\hat{k}=k/4$	$\hat{k}=k/2$	$\hat{k}=k$	$\hat{k}=2k$	$\hat{k}=4k$
$\varepsilon=1$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon=2$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.3 ± 0.5
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1.1 ± 1.6
$\varepsilon=3$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.3 ± 0.5
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1.1 ± 1.6
$\varepsilon=4$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon=5$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.7 ± 0.9	1.3 ± 1.2
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1.1 ± 1.6	4.8 ± 4.2
$\varepsilon=6$	precision	0.0 ± 0.0	0.0 ± 0.0	1.3 ± 1.9	4.0 ± 1.6	5.0 ± 0.8
	recall	0.0 ± 0.0	0.0 ± 0.0	1.1 ± 1.6	6.9 ± 2.6	18.0 ± 3.9
$\varepsilon=7$	precision	0.0 ± 0.0	5.1 ± 3.6	6.7 ± 6.8	8.0 ± 3.3	13.0 ± 4.5
	recall	0.0 ± 0.0	2.5 ± 1.8	6.0 ± 5.8	14.5 ± 6.1	46.8 ± 16.7
$\varepsilon=8$	precision	4.8 ± 6.7	15.4 ± 12.6	14.7 ± 13.2	18.0 ± 7.1	19.0 ± 5.7
	recall	1.4 ± 2.0	7.4 ± 5.7	13.4 ± 11.3	33.2 ± 14.8	68.8 ± 22.5
$\varepsilon=9$	precision	4.8 ± 6.7	12.8 ± 9.6	21.3 ± 15.4	29.3 ± 12.3	23.7 ± 2.4
	recall	1.4 ± 2.0	6.2 ± 4.5	20.1 ± 14.2	54.5 ± 25.9	84.5 ± 11.2
$\varepsilon=10$	precision	17.9 ± 5.1	32.6 ± 4.2	39.7 ± 11.7	42.7 ± 8.2	27.0 ± 4.1
	recall	4.6 ± 1.2	15.5 ± 2.1	36.7 ± 13.2	77.1 ± 19.2	94.9 ± 3.6

unconstrained		$\hat{k}=k/4$	$\hat{k}=k/2$	$\hat{k}=k$	$\hat{k}=2k$	$\hat{k}=4k$
$\varepsilon=1$	precision	1.1 ± 0.7	0.5 ± 0.4	0.7 ± 0.2	0.5 ± 0.1	0.5 ± 0.2
	recall	0.3 ± 0.2	0.3 ± 0.2	0.7 ± 0.1	0.9 ± 0.1	2.0 ± 0.5
$\varepsilon=2$	precision	2.6 ± 2.7	1.6 ± 1.3	1.9 ± 1.1	1.1 ± 0.5	0.6 ± 0.3
	recall	0.6 ± 0.6	0.8 ± 0.6	1.8 ± 1.1	2.2 ± 1.0	2.5 ± 1.1
$\varepsilon=3$	precision	2.6 ± 1.5	2.9 ± 1.0	3.7 ± 1.8	2.5 ± 1.1	1.2 ± 0.6
	recall	0.7 ± 0.3	1.5 ± 0.4	3.8 ± 1.8	4.9 ± 2.2	4.9 ± 2.2
$\varepsilon=4$	precision	7.9 ± 3.9	6.7 ± 2.6	9.7 ± 2.4	5.8 ± 1.7	3.0 ± 0.9
	recall	2.0 ± 0.8	3.3 ± 1.0	9.8 ± 1.6	11.7 ± 2.9	12.1 ± 2.9
$\varepsilon=5$	precision	10.1 ± 1.5	14.1 ± 3.3	20.4 ± 3.5	12.4 ± 1.7	6.5 ± 1.0
	recall	2.6 ± 0.5	7.2 ± 1.4	20.9 ± 2.6	25.2 ± 2.1	26.6 ± 1.6
$\varepsilon=6$	precision	19.0 ± 0.0	24.0 ± 3.5	38.3 ± 4.7	23.3 ± 3.7	12.3 ± 2.3
	recall	5.0 ± 0.6	12.5 ± 2.7	39.3 ± 3.4	47.4 ± 3.0	49.9 ± 4.5
$\varepsilon=7$	precision	30.2 ± 5.6	42.4 ± 8.5	56.9 ± 4.7	33.0 ± 4.9	17.3 ± 2.7
	recall	8.0 ± 2.4	22.2 ± 6.5	58.5 ± 2.2	67.1 ± 2.9	70.1 ± 3.5
$\varepsilon=8$	precision	49.2 ± 5.9	61.9 ± 5.9	69.3 ± 5.4	39.1 ± 5.1	19.9 ± 2.7
	recall	13.0 ± 2.9	32.1 ± 4.9	71.2 ± 2.4	79.8 ± 1.8	81.2 ± 1.9
$\varepsilon=9$	precision	63.5 ± 7.2	74.4 ± 4.1	76.1 ± 5.5	43.4 ± 5.9	22.2 ± 3.0
	recall	16.7 ± 3.6	38.5 ± 4.6	78.2 ± 3.1	88.4 ± 2.3	90.6 ± 2.2
$\varepsilon=10$	precision	79.9 ± 3.0	84.8 ± 1.7	81.5 ± 5.1	45.5 ± 5.6	23.1 ± 2.9
	recall	20.9 ± 3.1	43.9 ± 5.0	83.8 ± 4.3	92.8 ± 1.3	94.3 ± 1.3

high		$\hat{k}=k/4$	$\hat{k}=k/2$	$\hat{k}=k$	$\hat{k}=2k$	$\hat{k}=4k$
$\varepsilon=1$	precision	2.4 ± 0.5	2.0 ± 0.5	1.5 ± 0.4	1.4 ± 0.1	1.3 ± 0.1
	recall	0.5 ± 0.1	0.9 ± 0.2	1.3 ± 0.3	2.5 ± 0.1	4.6 ± 0.2
$\varepsilon=2$	precision	6.3 ± 1.5	3.8 ± 0.9	2.6 ± 0.5	1.8 ± 0.3	1.6 ± 0.2
	recall	1.4 ± 0.3	1.6 ± 0.4	2.2 ± 0.4	3.1 ± 0.5	5.7 ± 0.7
$\varepsilon=3$	precision	17.0 ± 3.4	9.1 ± 1.8	5.5 ± 0.9	3.6 ± 0.6	2.5 ± 0.3
	recall	3.7 ± 0.6	3.9 ± 0.7	4.7 ± 0.6	6.1 ± 0.8	8.6 ± 1.0
$\varepsilon=4$	precision	35.9 ± 3.2	23.5 ± 2.3	13.1 ± 1.5	7.6 ± 0.7	5.0 ± 0.6
	recall	7.8 ± 0.5	10.2 ± 0.7	11.3 ± 0.8	13.2 ± 0.8	17.1 ± 1.6
$\varepsilon=5$	precision	56.8 ± 0.4	53.5 ± 2.8	29.1 ± 1.7	17.1 ± 0.8	10.7 ± 0.5
	recall	12.3 ± 0.4	23.2 ± 0.3	25.3 ± 0.6	29.7 ± 0.5	37.2 ± 0.5
$\varepsilon=6$	precision	74.8 ± 0.3	79.2 ± 0.9	55.8 ± 1.7	31.7 ± 0.9	18.3 ± 0.8
	recall	16.2 ± 0.6	34.4 ± 1.5	48.4 ± 0.6	54.9 ± 0.8	63.4 ± 0.8
$\varepsilon=7$	precision	86.6 ± 1.2	88.8 ± 0.4	79.8 ± 3.8	43.1 ± 2.2	23.4 ± 1.1
	recall	18.8 ± 1.0	38.6 ± 1.6	69.2 ± 1.0	74.7 ± 1.1	81.0 ± 0.6
$\varepsilon=8$	precision	92.5 ± 1.1	93.2 ± 0.8	90.1 ± 2.3	49.1 ± 2.3	26.0 ± 1.2
	recall	20.1 ± 1.0	40.5 ± 1.6	78.2 ± 1.5	85.2 ± 0.7	90.0 ± 0.5
$\varepsilon=9$	precision	95.9 ± 0.7	96.4 ± 0.5	94.3 ± 0.7	52.8 ± 2.4	27.3 ± 1.3
	recall	20.8 ± 0.9	41.9 ± 1.8	81.9 ± 2.8	91.5 ± 0.6	94.7 ± 0.8
$\varepsilon=10$	precision	97.4 ± 0.3	97.8 ± 0.3	95.5 ± 0.2	54.6 ± 2.2	27.8 ± 1.2
	recall	21.2 ± 0.8	42.5 ± 1.8	82.9 ± 3.1	94.6 ± 0.2	96.5 ± 0.2

TABLE XXV: Flickr (EDGERAND)

low		$\hat{k} = k/4$	$\hat{k} = k/2$	$\hat{k} = k$	$\hat{k} = 2k$	$\hat{k} = 4k$
$\varepsilon = 5$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon = 6$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	2.6 ± 3.6	1.3 ± 1.9
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	5.6 ± 7.9	5.6 ± 7.9
$\varepsilon = 7$	precision	16.7 ± 23.6	8.3 ± 11.8	4.8 ± 6.7	2.6 ± 3.6	1.3 ± 1.9
	recall	5.6 ± 7.9	5.6 ± 7.9	5.6 ± 7.9	5.6 ± 7.9	5.6 ± 7.9
$\varepsilon = 8$	precision	16.7 ± 23.6	8.3 ± 11.8	4.8 ± 6.7	8.5 ± 6.4	4.3 ± 3.3
	recall	5.6 ± 7.9	5.6 ± 7.9	5.6 ± 7.9	17.8 ± 13.7	17.8 ± 13.7
$\varepsilon = 9$	precision	33.3 ± 47.1	16.7 ± 23.6	28.1 ± 15.7	20.1 ± 3.9	10.4 ± 2.2
	recall	8.3 ± 11.8	8.3 ± 11.8	28.9 ± 15.0	41.1 ± 6.8	41.1 ± 6.8
$\varepsilon = 10$	precision	66.7 ± 47.1	38.9 ± 28.3	37.9 ± 15.7	19.3 ± 7.7	12.9 ± 5.5
	recall	24.4 ± 17.5	24.4 ± 17.5	39.4 ± 14.9	39.4 ± 14.9	51.7 ± 22.5

unconstrained		$\hat{k} = k/4$	$\hat{k} = k/2$	$\hat{k} = k$	$\hat{k} = 2k$	$\hat{k} = 4k$
$\varepsilon = 5$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon = 6$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1.3 ± 1.9	0.7 ± 0.9
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	2.2 ± 3.1	2.2 ± 3.1
$\varepsilon = 7$	precision	0.0 ± 0.0	0.0 ± 0.0	2.6 ± 3.6	2.7 ± 1.9	1.3 ± 0.9
	recall	0.0 ± 0.0	0.0 ± 0.0	2.4 ± 3.4	4.6 ± 3.3	4.6 ± 3.3
$\varepsilon = 8$	precision	0.0 ± 0.0	4.8 ± 6.7	2.6 ± 3.6	4.0 ± 5.7	3.5 ± 1.8
	recall	0.0 ± 0.0	2.4 ± 3.4	2.4 ± 3.4	7.1 ± 10.1	12.7 ± 6.3
$\varepsilon = 9$	precision	8.3 ± 11.8	9.5 ± 6.7	13.6 ± 2.5	11.3 ± 0.9	7.2 ± 0.8
	recall	2.2 ± 3.1	4.6 ± 3.3	12.5 ± 1.8	20.5 ± 0.7	26.2 ± 4.4
$\varepsilon = 10$	precision	27.8 ± 3.9	21.0 ± 5.9	17.7 ± 9.3	18.0 ± 10.7	11.0 ± 3.7
	recall	7.9 ± 1.5	10.3 ± 3.1	17.0 ± 9.7	34.1 ± 21.8	41.0 ± 16.4

high		$\hat{k} = k/4$	$\hat{k} = k/2$	$\hat{k} = k$	$\hat{k} = 2k$	$\hat{k} = 4k$
$\varepsilon = 5$	precision	14.9 ± 4.9	9.8 ± 3.2	6.1 ± 1.8	4.1 ± 0.6	2.6 ± 0.3
	recall	3.9 ± 1.6	5.1 ± 2.1	6.3 ± 2.4	8.2 ± 2.0	10.6 ± 2.0
$\varepsilon = 6$	precision	32.4 ± 4.2	26.8 ± 3.3	16.8 ± 2.1	11.0 ± 1.4	7.2 ± 0.5
	recall	8.2 ± 1.7	13.6 ± 3.0	17.1 ± 3.8	22.2 ± 5.1	29.1 ± 4.7
$\varepsilon = 7$	precision	38.1 ± 1.7	34.7 ± 2.6	23.3 ± 3.7	15.7 ± 1.8	9.9 ± 0.7
	recall	9.6 ± 1.4	17.6 ± 3.0	23.7 ± 5.9	31.5 ± 5.7	39.8 ± 6.4
$\varepsilon = 8$	precision	34.5 ± 3.1	35.2 ± 2.9	25.6 ± 2.8	16.5 ± 1.9	10.5 ± 0.4
	recall	8.8 ± 1.7	17.9 ± 3.3	25.8 ± 4.7	33.4 ± 6.4	42.3 ± 5.5
$\varepsilon = 9$	precision	24.6 ± 3.8	32.0 ± 5.4	24.2 ± 2.9	14.9 ± 1.4	9.7 ± 0.5
	recall	6.3 ± 1.6	16.4 ± 4.4	24.5 ± 5.0	30.0 ± 5.3	38.9 ± 5.3
$\varepsilon = 10$	precision	19.6 ± 4.2	25.3 ± 6.2	20.6 ± 3.1	14.4 ± 2.1	9.7 ± 0.9
	recall	5.0 ± 1.5	13.0 ± 4.4	20.9 ± 5.0	29.0 ± 6.5	39.1 ± 7.5

TABLE XXVI: Flickr (LAPGRAPH)

low		$\hat{k} = k/4$	$\hat{k} = k/2$	$\hat{k} = k$	$\hat{k} = 2k$	$\hat{k} = 4k$
$\varepsilon = 1$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon = 2$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon = 3$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon = 4$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon = 5$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon = 6$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon = 7$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon = 8$	precision	0.0 ± 0.0	0.0 ± 0.0	6.7 ± 9.4	3.3 ± 4.7	6.1 ± 5.5
	recall	0.0 ± 0.0	0.0 ± 0.0	6.7 ± 9.4	6.7 ± 9.4	23.3 ± 20.5
$\varepsilon = 9$	precision	0.0 ± 0.0	11.1 ± 15.7	26.4 ± 10.5	13.4 ± 5.1	6.9 ± 2.5
	recall	0.0 ± 0.0	6.7 ± 9.4	27.2 ± 9.7	27.2 ± 9.7	27.2 ± 9.7
$\varepsilon = 10$	precision	0.0 ± 0.0	11.1 ± 15.7	16.2 ± 12.0	21.9 ± 7.2	14.8 ± 1.1
	recall	0.0 ± 0.0	6.7 ± 9.4	17.8 ± 13.7	45.0 ± 14.7	58.9 ± 6.8

unconstrained		$\hat{k} = k/4$	$\hat{k} = k/2$	$\hat{k} = k$	$\hat{k} = 2k$	$\hat{k} = 4k$
$\varepsilon = 1$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon = 2$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon = 3$	precision	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	recall	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
$\varepsilon = 4$	precision	0.0 ± 0.0	4.8 ± 6.7	2.6 ± 3.6	1.3 ± 1.9	0.7 ± 0.9
	recall	0.0 ± 0.0	2.2 ± 3.1	2.2 ± 3.1	2.2 ± 3.1	2.2 ± 3.1
$\varepsilon = 5$	precision	0.0 ± 0.0	0.0 ± 0.0	2.6 ± 3.6	1.3 ± 1.9	1.3 ± 0.9
	recall	0.0 ± 0.0	0.0 ± 0.0	2.2 ± 3.1	2.2 ± 3.1	4.6 ± 3.3
$\varepsilon = 6$	precision	0.0 ± 0.0	4.8 ± 6.7	2.6 ± 3.6	1.3 ± 1.9	1.3 ± 0.9
	recall	0.0 ± 0.0	2.2 ± 3.1	2.2 ± 3.1	2.2 ± 3.1	4.6 ± 3.3
$\varepsilon = 7$	precision	0.0 ± 0.0	0.0 ± 0.0	2.6 ± 3.6	4.0 ± 3.3	2.0 ± 1.6
	recall	0.0 ± 0.0	0.0 ± 0.0	2.2 ± 3.1	6.8 ± 5.4	6.8 ± 5.4
$\varepsilon = 8$	precision	0.0 ± 0.0	0.0 ± 0.0	2.6 ± 3.6	5.3 ± 5.0	3.3 ± 2.5
	recall	0.0 ± 0.0	0.0 ± 0.0	2.2 ± 3.1	9.0 ± 8.3	11.4 ± 8.4
$\varepsilon = 9$	precision	8.3 ± 11.8	4.8 ± 6.7	2.6 ± 3.6	5.3 ± 5.0	4.0 ± 3.3
	recall	2.2 ± 3.1	2.2 ± 3.1	2.2 ± 3.1	9.4 ± 9.0	14.0 ± 11.7
$\varepsilon = 10$	precision	19.4 ± 14.2	16.2 ± 12.0	14.4 ± 5.1	10.0 ± 1.6	7.0 ± 3.6
	recall	5.6 ± 4.2	7.8 ± 5.7	13.5 ± 5.3	18.3 ± 3.5	25.4 ± 12.6

high		$\hat{k} = k/4$	$\hat{k} = k/2$	$\hat{k} = k$	$\hat{k} = 2k$	$\hat{k} = 4k$
$\varepsilon = 1$	precision	0.4 ± 0.5	0.2 ± 0.3	0.1 ± 0.1	0.0 ± 0.1	0.0 ± 0.0
	recall	0.1 ± 0.1	0.1 ± 0.1	0.1 ± 0.1	0.1 ± 0.1	0.1 ± 0.1
$\varepsilon = 2$	precision	0.4 ± 0.5	0.2 ± 0.3	0.1 ± 0.1	0.0 ± 0.1	0.0 ± 0.0
	recall	0.1 ± 0.1	0.1 ± 0.1	0.1 ± 0.1	0.1 ± 0.1	0.1 ± 0.1
$\varepsilon = 3$	precision	0.4 ± 0.5	0.2 ± 0.3	0.1 ± 0.1	0.0 ± 0.1	0.0 ± 0.0
	recall	0.1 ± 0.1	0.1 ± 0.1	0.1 ± 0.1	0.1 ± 0.1	0.1 ± 0.1
$\varepsilon = 4$	precision	1.6 ± 1.3	1.0 ± 0.9	0.6 ± 0.5	0.3 ± 0.2	0.2 ± 0.1
	recall	0.4 ± 0.4	0.5 ± 0.5	0.6 ± 0.5	0.6 ± 0.5	0.6 ± 0.5
$\varepsilon = 5$	precision	3.5 ± 1.3	2.0 ± 0.9	1.2 ± 0.5	0.6 ± 0.3	0.3 ± 0.1
	recall	0.9 ± 0.4	1.0 ± 0.5	1.2 ± 0.5	1.2 ± 0.5	1.2 ± 0.5
$\varepsilon = 6$	precision	11.5 ± 0.9	7.9 ± 0.5	5.0 ± 0.7	2.5 ± 0.4	1.3 ± 0.2
	recall	2.9 ± 0.5	4.0 ± 0.7	5.0 ± 0.3	5.0 ± 0.3	5.0 ± 0.3
$\varepsilon = 7$	precision	18.2 ± 2.2	14.8 ± 0.7	11.4 ± 1.4	7.6 ± 1.2	3.9 ± 0.7
	recall	4.6 ± 0.9	7.4 ± 0.8	11.6 ± 2.5	15.0 ± 1.2	15.2 ± 1.3
$\varepsilon = 8$	precision	19.6 ± 0.7	16.9 ± 0.6	14.1 ± 1.1	11.9 ± 0.8	9.0 ± 1.1
	recall	4.9 ± 0.4	8.5 ± 1.3	14.2 ± 2.2	24.0 ± 3.3	35.7 ± 2.1
$\varepsilon = 9$	precision	15.9 ± 1.7	17.8 ± 3.2	15.6 ± 1.7	12.9 ± 1.3	10.6 ± 1.0
	recall	4.0 ± 0.7	9.1 ± 2.5	15.9 ± 3.4	26.2 ± 5.4	43.1 ± 8.5
$\varepsilon = 10$	precision	14.5 ± 3.3	19.7 ± 3.7	16.9 ± 2.1	13.1 ± 1.4	10.8 ± 1.0
	recall	3.7 ± 1.1	10.0 ± 2.5	17.1 ± 3.4	26.5 ± 5.5	43.6 ± 8.1