Locally Private Graph Neural Networks

Sina Sajadmanesh
sajadmanesh@idiap.ch
Idiap Research Institute
EPFL

Daniel Gatica-Perez
gatica@idiap.ch
Idiap Research Institute
EPFL

ABSTRACT

Graph Neural Networks (GNNs) have demonstrated superior performance in learning node representations for various graph inference tasks. However, learning over graph data can raise privacy concerns when nodes represent people or human-related variables that involve sensitive or personal information. While numerous techniques have been proposed for privacy-preserving deep learning over non-relational data, there is less work addressing the privacy issues pertained to applying deep learning algorithms on graphs. In this paper, we study the problem of node data privacy, where graph nodes have potentially sensitive data that is kept private, but they could be beneficial for a central server for training a GNN over the graph. To address this problem, we develop a privacy-preserving, architecture-agnostic GNN learning algorithm with formal privacy guarantees based on Local Differential Privacy (LDP). Specifically, we propose an LDP encoder and an unbiased rectifier, by which the server can communicate with the graph nodes to privately collect their data and approximate the GNN's first layer. To further reduce the effect of the injected noise, we propose to prepend a simple graph convolution layer, called KProp, which is based on the multi-hop aggregation of the nodes' features acting as a denoising mechanism. Finally, we propose a robust training framework, in which we benefit from KProp's denoising capability to increase the accuracy of inference in the presence of noisy labels. Extensive experiments conducted over real-world datasets demonstrate that our method can maintain a satisfying level of accuracy with low privacy loss.

1 INTRODUCTION

In the past few years, extending deep learning models for graph-structured data has attracted growing interest, popularizing the concept of Graph Neural Networks (GNNs) [44]. GNNs have shown superior performance in a wide range of applications in social sciences [18], biology [41], molecular chemistry [14], and so on, achieving state-of-the-art results in various graph-based learning tasks, such as node classification [26], link prediction [65], and community detection [9]. However, most real-world graphs associated with people or human-related activities, such as social and economic networks, are often sensitive and might contain personal information. For example in a social network, a user's friend list, profile information, likes and comments, etc., could potentially be private to the user. To satisfy users' privacy expectations in accordance with recent legal data protection policies, it is of great importance to develop privacy-preserving GNN models for applications that rely on graphs accessing users' personal data.

Problem and motivation. In light of these privacy constraints, we define the problem of node data privacy. As illustrated in Figure 1, in this setting, graph nodes, which may represent human users, have potentially sensitive data in the form of feature vectors and possibly labels that are kept private, but the topology of the graph is observable from the viewpoint of a central server, whose goal is to benefit from private node data to learn a GNN over the graph. This problem has many applications in social network analysis and mobile computing. For example, consider a social smartphone application server, e.g., a social network, messaging platform, or a dating app. As this server already has the data about social interactions between its users, the graph topology is not private to the server. However, the server could potentially benefit from users' personal information, such as their phone's sensor data, list of installed apps, or application usage logs, by training a GNN using these private features to learn better user representations for improving its services (e.g., the recommendation system). Without any means of data protection, however, this implies that the server should collect users' personal data directly, which can raise privacy concerns.

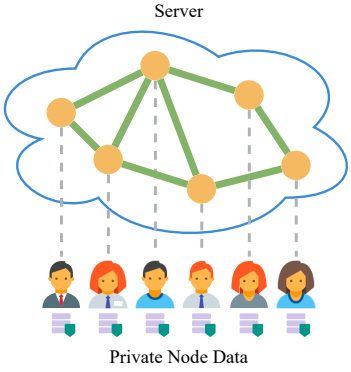


Figure 1: The node data privacy problem. A cloud server (e.g., a social network server) has a graph (e.g., the social graph), whose nodes, which may correspond to real users, have some private data that the server wishes to utilize for training a GNN on the graph, but cannot simply collect them due to privacy constraints.

Challenges. Training a GNN from private node data is a challenging task, mainly due to the relational nature of graphs. Unlike other deep learning models wherein the training data points are independent, in GNNs, the samples – nodes of the graph – are connected via links and exchange information through the message-passing framework during training [19]. This fact renders common collaborative learning paradigms, such as federated learning [23], infeasible due to their excessive communication overhead. The main reason is that in the absence of a trusted server, which is the primary assumption of our paper, every adjacent pair of nodes has to exchange their vector representations with each other multiple

times during a single training epoch of a GNN, which requires significantly more communication compared to conventional deep neural networks, where the nodes only communicate with the server, independently.

Contributions. In this paper, we propose the Locally Private Graph Neural Network (LPGNN), a novel privacy-preserving GNN learning framework for training GNN models using private node data. Our method has provable privacy guarantees based on Local Differential Privacy (LDP) [24], can be used when either or both node features and labels are private, and can be combined with any GNN architecture independently.

To protect the privacy of node features, we propose an LDP mechanism, called the *multi-bit mechanism*, through which the graph nodes can perturb their features that are then collected by the server with minimum communication overhead. These noisy features are then used to estimate the first graph convolution layer of the GNN. Given that graph convolution layers initially aggregate node features before passing them through non-linear activation functions, we benefit from this aggregation step as a denoising mechanism to average out the differentially private noise we have injected into the node features. To further improve the effectiveness of this denoising process and increase the estimation accuracy of the graph convolution, we propose to prepend a simple yet effective graph convolution layer based on the multi-hop aggregation of node features, called KProp, to the backbone GNN.

To preserve the privacy of node labels, we perturb them using the generalized randomized response mechanism [22]. However, learning with perturbed labels introduces extra challenges, as the label noise could significantly degrade the generalization performance of the GNN. To this end, we propose a robust training framework, called Drop (label **d**enoising with p**rop**agation), in which we again benefit from KProp's denoising capability to increase the accuracy of noisy labels. Drop can be seamlessly combined with any GNN, is very easy to train, and does not rely on any clean (raw) data in any form, being features or labels, neither for training nor validation and hyper-parameter optimization.

Finally, we derive the theoretical properties of the proposed algorithms, including the formal privacy guarantees and error bound. We conduct extensive experiments over several real-world datasets, which demonstrate that our proposed LPGNN is robust against injected LDP noise, achieving a decent accuracy-privacy trade-off in the presence of noisy features and labels.

Paper organization. The rest of this paper is organized as follows. In Section 2, we formally define the problem and provide the necessary backgrounds. Then, in Section 3, we explain our locally private GNN training algorithm. Details of experiments and their results are explained in Section 4. We review related work in Section 5 and finally in Section 6, we conclude the paper. The proofs of all the theoretical findings are also presented in Appendix A.

2 PRELIMINARIES

Problem definition. We formally define the problem of learning a GNN with node data privacy. Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X}, \mathbf{Y})$, where $\mathcal{E}$ is the link set and $\mathcal{V} = \mathcal{V}_{\mathcal{L}} \cup \mathcal{V}_{\mathcal{U}}$ is the union of the set of labeled nodes $\mathcal{V}_{\mathcal{L}}$ and unlabeled ones $\mathcal{V}_{\mathcal{U}}$. The feature matrix $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$ comprises d-dimensional feature vectors $\mathbf{x}_v$ for each node $v \in \mathcal{V}$, and $\mathbf{Y} \in \{0,1\}^{|\mathcal{V}| \times c}$ is the label matrix, where c is the number of classes. For each node $v \in \mathcal{V}_{\mathcal{L}}$, $\mathbf{y}_v$ is a one-hot vector, i.e., $\mathbf{y}_v \cdot \vec{\mathbf{1}} = 1$, where $\vec{\mathbf{1}}$ is the all-one vector, and for each node $v \in \mathcal{V}_{\mathcal{U}}$, $\mathbf{y}_v$ is the all-zero vector $\vec{\mathbf{0}}$. Now assume that a server has access to $\mathcal{V}$ and $\mathcal{E}$, but the feature matrix $\mathbf{X}$ and labels $\mathbf{Y}$ are private to the nodes and thus not observable by the server. The problem is: how can the server collaborate with the nodes to train a GNN over $\mathcal{G}$ without letting private data leave the nodes? To answer this question, we first present the required background about graph neural networks and local differential privacy in the following, and then in the next section, we describe our proposed method in detail. Note that since in our problem setting, nodes of the graph usually correspond to human users, we often use the terms "node" and "user" interchangably throughout the rest of the paper.

Graph Neural Networks. A GNN learns a representation for every node in the graph using a set of stacked graph convolution layers. Each layer gets an initial vector for each node and outputs a new embedding vector by aggregating the vectors of the adjacent neighbors followed by a non-linear transformation. More formally, given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, an L-layer GNN consists of L graph convolution layers, where the embedding $\mathbf{h}_v^l$ of any node $v \in \mathcal{V}$ at layer l is generated by aggregating the previous layer's embeddings of its neighbors, called the neighborhood aggregation step, as:

$$\mathbf{h}_{\mathcal{N}(v)}^l = \textsc{Aggregate}_l\left(\{\mathbf{h}_u^{l-1}, \forall u \in \mathcal{N}(v)\}\right) \tag{1}$$

$$\mathbf{h}_v^l = \textsc{Update}_l\left(\mathbf{h}_{\mathcal{N}(v)}^l\right) \tag{2}$$

where $\mathcal{N}(v)$ is the set of neighbors of v (which could include v itself) and $\mathbf{h}_u^{l-1}$ is the embedding of node u at layer $l-1$. $\textsc{Aggregate}_l(.)$ and $\mathbf{h}_{\mathcal{N}(v)}^l$ are respectively the l-th layer differentiable, permutation invariant aggregator function (such as mean, sum, or max) and its output on $\mathcal{N}(v)$. Finally, $\textsc{Update}_l(.)$ is a trainable non-linear function, e.g., a neural network, for layer l. At the very first, we have $\mathbf{h}_v^0 = \mathbf{x_v}$, i.e., the initial embedding of v is its feature vector $\mathbf{x}_v$, and the last layer generates a c-dimensional output followed by a softmax layer to predict node labels in a c-class node classification task.

Local Differential Privacy. Local differential privacy (LDP) is an increasingly used approach for collecting private data and computing statistical queries, such as mean, count, and histogram. It has been already deployed by major technology companies, including Google [16], Apple [46], and Microsoft [11]. The key idea behind LDP is that data holders do not need to share their private data with an untrusted data aggregator, but instead send a perturbed version of their data, which is not meaningful individually but can approximate the target query when aggregated. It includes two steps: (i) data holders perturb their data using a special randomized mechanism $\mathcal{M}$ and send the output to the aggregator; and (ii) the aggregator combines all the received perturbed values and estimates the target query. To prevent the aggregator from inferring the original private value from the perturbed one, the mechanism $\mathcal{M}$ must satisfy the following definition [24]:

Definition 2.1. Given $\epsilon > 0$, a randomized mechanism $\mathcal{M}$ satisfies ϵ-local differential privacy, if for all possible pairs of user's private

data x and x', and for all possible outputs $y \in Range(\mathcal{M})$, we have:

$$\Pr[\mathcal{M}(x) = y] \leq e^{\epsilon} \Pr[\mathcal{M}(x') = y] \quad (3)$$

The parameter ϵ in the above definition is called the *"privacy budget"* and is used to tune utility versus privacy: a smaller (resp. larger) ϵ leads to stronger (resp. weaker) privacy guarantees, but lower (resp. higher) utility. The above definition implies that the mechanism $\mathcal{M}$ should assign similar probabilities (controlled by ϵ) to the outputs of different input values x and x', so that by looking at the outputs, an adversary could not infer the input value with high probability, regardless of any side knowledge they might have. LDP is achieved for a deterministic function usually by adding a special random noise to its output that cancels out when calculating the target aggregation given a sufficiently large number of noisy samples.

3 PROPOSED METHOD

In this section, we describe our proposed framework for learning a GNN using private node data. As described in the previous section, in the forward propagation of a GNN, the node features are only used as the input to the first layer's Aggregate function. This aggregation step is amenable to privacy, as it allows us to perturb node features using an LDP mechanism (e.g., by injecting random noise into the features) and then let the Aggregate function average out the injected noise (to an extent, not entirely), yielding a relatively good approximation of the neighborhood aggregation for the subsequent Update function. The GNN's forward propagation can then proceed from this point without any modification to predict a class label for each node.

However, maintaining a proper balance between the accuracy of the GNN and the privacy of data introduces new challenges that need to be carefully addressed. On one hand, the node features to be collected are likely high-dimensional, so the perturbation of every single feature consumes a lot of the privacy budget. Suppose we want to keep our total budget ϵ low to provide better privacy protection. In that case, we need to perturb each of the d features with ϵ/d budget (because the privacy budgets of the features add up together as the result of the composition theorem [15]), which in turn results in adding more noise to the data that can significantly degrade the final accuracy. On the other hand, for the GNN to be able to cancel out the injected noise, the first layer's aggregator function must: (i) be in the form of a linear summation, and (ii) be calculated over a sufficiently large set of node features. However, not every GNN architecture employs a linear aggregator function, nor every node in the graph has many neighbors. In fact, in many real-world graphs that follow a Power-Law degree distribution, the number of low-degree nodes is much higher than the high-degree ones. Consequently, the estimated aggregation would most likely be very noisy, again leading to degraded performance.

To tackle the first challenge, we develop a multidimensional LDP method, called the *multi-bit mechanism*, by extending the 1-bit mechanism [11] for multidimensional feature collection. It is composed of a user-side encoder and a server-side rectifier designed for maximum communication efficiency. To address the second challenge, we propose a simple, yet effective graph convolution layer, called *KProp*, which aggregates messages from an expanded neighborhood set that includes both the immediate neighbors and those nodes that are up to K-hops away. By prepending this layer to the GNN, we can both combine our method with any GNN architecture and at the same time increase the graph convolution's estimation accuracy for low-degree nodes. In the experiments, we show that this technique can significantly boost the performance of our locally private GNN, especially for graphs with a lower average degree.

Finally, since the node labels are also considered private, we need another LDP mechanism to collect them privately. To this end, we use the generalized randomized response algorithm [22], which randomly flips the correct label to another one with a probability that depends on the privacy budget. However, learning the GNN with perturbed labels brings forward significant challenges in both training and validation. Regarding the former, training the GNN directly with the perturbed labels causes the model to overfit the noisy labels, leading to poor generalization performance. Regarding the latter, while it would be easy to validate the trained model using clean (non-perturbed) data, due to the privacy constraints of our problem, a more realistic setting is to assume that the server does not have access to any clean validation data. In this case, it is not clear how to perform model validation with noisy data, which is vital to prevent overfitting and optimize model hyper-parameters.

Although deep learning with noisy labels has been studied extensively in the literature [31, 38, 39, 45, 63, 64, 67], almost all the previous works either need clean features for training, require clean data for validation, or have been proposed for standard deep neural networks and do not consider the graph structure. Here, we propose *Label **D**enoising with P**rop**agation – Drop*, which incorporates the graph structure for label correction, and at the same time does not rely on any form of clean data (features or labels), neither for training nor validation. Given that nodes with similar labels tend to connect together more often [48], we utilize the graph topology to predict the label of a node by estimating the label frequency of its neighboring nodes. Still, if we rely on immediate neighbors, the true labels could not be accurately estimated due to insufficient neighbors for many nodes. Again, our key idea is to exploit KProp's denoising capability, but this time on node labels, to estimate the label frequency for each node and recovering the true label by choosing the most frequent one. Drop can easily be combined by any GNN architecture, and we show that it outperforms traditional baselines, especially at high-privacy regimes.

In the rest of this section, we describe our multi-bit mechanism, the KProp layer, and the Drop algorithms in more detail. The overview of our framework is depicted in Figure 2. Note that the data perturbation step on the user-side has to be done only once for each node. The server collects the perturbed data once and stores it to train the GNN with minimum communication overhead.

3.1 Collection of node features

In this section, we explain our multi-bit mechanism for multidimensional feature perturbation, which is composed of an encoder and a rectifier, as described in the following.

Multi-bit Encoder. This part, which is executed at the user-side, perturbs the node's private feature vector and encodes it into a compact vector that is sent efficiently to the server. More specifically, assume that every node v owns a private d-dimensional feature

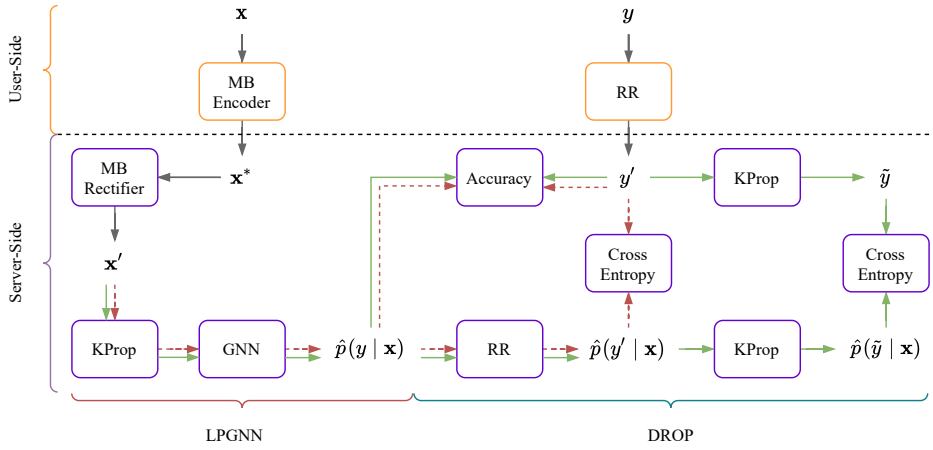


Figure 2: Overview of our locally private GNN training framework, featuring the multi-bit mechanism (MB Encoder and MB Rectifier), randomized response (RR), KProp layers, and Drop training. Users run multi-bit encoder and randomized response on their private features and labels, respectively, and send the output to the server, after which training begins. Green solid arrows and red dashed arrows indicate the training and validation paths, respectively.

Algorithm 1: Multi-Bit Encoder

Input : feature vector $\mathbf{x} \in [\alpha, \beta]^d$; privacy budget $\epsilon > 0$; range parameters α and β; sampling parameter $m \in \{1, 2, \ldots, d\}$.

Output : encoded vector $\mathbf{x}^* \in \{-1, 0, 1\}^d$.

Let $\mathcal{S}$ be a set of m values drawn uniformly at random without replacement from $\{1, 2, \ldots, d\}$

for $i \in \{1, 2, \ldots, d\}$ **do**

  $s_i = 1$ if $i \in \mathcal{S}$ otherwise $s_i = 0$

  $t_i \sim \text{Bernoulli}\left(\frac{1}{e^{\epsilon/m}+1} + \frac{x_i - \alpha}{\beta - \alpha} \cdot \frac{e^{\epsilon/m}-1}{e^{\epsilon/m}+1}\right)$

  $x_i^* = s_i \cdot (2t_i - 1)$

end

return $\mathbf{x}^* = [x_1^*, \ldots, x_d^*]^T$

vector $\mathbf{x}_v$, whose elements lie in the range $[\alpha, \beta]$. When the server requests the feature vector of v, the node locally applies the multi-bit encoder on $\mathbf{x}_v$ to get the corresponding encoded feature vector $\mathbf{x}_v^*$, which is then sent back to the server. Since this process is supposed to be run only once, the generated $\mathbf{x}_v^*$ is recorded by the node to be returned in any subsequent calls to prevent the server from recovering the private feature vector using repeated queries.

Our multi-bit encoder is built upon the 1-bit mechanism [11], which returns either 0 or 1 for a single-dimensional input. However, as mentioned earlier, perturbing all the dimensions with a high-dimensional input results in injecting too much noise, as the total privacy budget has to be shared among all the dimensions. To balance the privacy-accuracy trade-off, we need to reduce dimensionality to decrease the number of dimensions that have to be perturbed. Still, since we cannot have the feature vectors of all the nodes at one place (due to privacy reasons), we cannot use conventional approaches, such as principal component analysis (PCA) or any other machine learning-based feature selection method. Instead, we randomly perturb a subset of the dimensions and then optimize the size of this subset to achieve the lowest variance in estimating the AGGREGATE function.

Algorithm 1 describes this encoding process in greater detail. Intuitively, the encoder first uniformly samples m out of d dimensions without replacement, where m is a parameter controlling how many dimensions are perturbed. Then, for each sampled dimension, the corresponding feature is randomly mapped to either -1 or 1, with a probability depending on the per-dimension privacy budget ϵ/m and the position of the feature value in the feature space, such that values closer to α (resp. β) are likely to be mapped to -1 (resp. 1). For other dimensions that are not sampled, the algorithm outputs 0. Therefore, a maximum of two bits per feature is enough to send $\mathbf{x}_v^*$ to the server. When $m = d$, our algorithm reduces to the 1-bit mechanism with a privacy budget of ϵ/d for every single dimension. The following theorem ensures that the multi-bit encoder is ϵ-LDP (proof in the Appendix).

THEOREM 3.1. *The multi-bit encoder presented in Algorithm 1 satisfies ϵ-local differential privacy for each node.*

Multi-bit Rectifier. The output of the multi-bit encoder is statistically biased, i.e., $\mathbb{E}[\mathbf{x}^*] \neq \mathbf{x}$. Therefore, the goal of the multi-bit rectifier, executed at server-side, is to convert the encoded vector $\mathbf{x}^*$ to an unbiased perturbed vector $\mathbf{x}'$, such that $\mathbb{E}[\mathbf{x}'] = \mathbf{x}$, as follows:

$$\mathbf{x}' = Rect(\mathbf{x}^*) = \frac{d(\beta - \alpha)}{2m} \cdot \frac{e^{\epsilon/m} + 1}{e^{\epsilon/m} - 1} \cdot \mathbf{x}^* + \frac{\alpha + \beta}{2} \tag{4}$$

Note that this is not a denoising process to remove the noise from $\mathbf{x}^*$, but the output vector $\mathbf{x}'$ is still noisy and does not have any meaningful information about the private vector $\mathbf{x}$. The only difference between $\mathbf{x}^*$ and $\mathbf{x}'$ is that the latter is unbiased, while the former is not. The following results entail from the multi-bit rectifier:

PROPOSITION 3.2. *The multi-bit rectifier defined by (4) is unbiased.*

Proposition 3.3. *For any node v and any $i \in \{1, 2, \ldots, d\}$, the variance of the multi-bit rectifier defined by (4) at dimension i is:*

$$Var[x'_{v,i}] = \frac{d}{m} \cdot \left(\frac{\beta - \alpha}{2} \cdot \frac{e^{\epsilon/m} + 1}{e^{\epsilon/m} - 1} \right)^2 - \left(x_{v,i} - \frac{\alpha + \beta}{2} \right)^2 \quad (5)$$

The variance of an LDP mechanism is a key factor affecting the estimation accuracy: a lower variance usually leads to a more accurate estimation. Therefore, we exploit the result of Proposition 3.3 to find the optimal sampling parameter m in the multi-bit encoder (Algorithm 1) that minimizes the rectifier's variance, as follows:

Proposition 3.4. *The optimal value of the sampling parameter m in Algorithm 1, denoted by $m^\star$, is obtained as:*

$$m^\star = \max(1, \min(d, \left\lfloor \frac{\epsilon}{2.18} \right\rfloor)) \quad (6)$$

The above proposition implies that in the high-privacy regime $\epsilon \leq 2.18$, the multi-bit mechanism perturbs only one random dimension. Therefore, this process is similar to a randomized one-hot encoding, except that here, the aggregation of these one-hot encoded features approximates the aggregation of the raw features.

3.2 Approximation of graph convolution

Upon collecting the encoded vectors $\mathbf{x}^*_v$ from every node v and generating the corresponding perturbed vectors $\mathbf{x}'_v$ using the multi-bit rectifier, the server can initiate the training of the GNN. In the first layer, the embedding for an arbitrary node v is generated by the following (layer indicator subscripts and superscripts are omitted for simplicity):

$$\widehat{\mathbf{h}}_{\mathcal{N}(v)} = \textsc{Aggregate}\left(\{\mathbf{x}'_u, \forall u \in \mathcal{N}(v)\}\right) \quad (7)$$

$$\mathbf{h}_v = \textsc{Update}\left(\widehat{\mathbf{h}}_{\mathcal{N}(v)}\right) \quad (8)$$

where $\widehat{\mathbf{h}}_{\mathcal{N}(v)}$ is the estimation of the first layer Aggregate function of any node v by aggregating perturbed vectors $\mathbf{x}'_u$ of all the nodes u adjacent to v. After this step, the server can proceed with the rest of the layers to complete the forward propagation of the model, exactly similar to a standard GNN. If the Aggregate function is linear on its input (e.g., it is a weighted summation of the input vectors), the resulting aggregation would also be unbiased, as stated below:

Corollary 3.5. *Given a linear aggregator function, the aggregation defined by (7) is an unbiased estimation for (1) at layer $l = 1$.*

The following proposition also shows the relationship of the estimation error in calculating the Aggregate function and the neighborhood size $|\mathcal{N}(v)|$ for the special case of using the mean aggregator function:

Proposition 3.6. *Given the mean aggregator function for the first layer and $\delta > 0$, with probability at least $1 - \delta$, for any node v, we have:*

$$\max_{i \in \{1,\ldots,d\}} \left| (\widehat{\mathbf{h}}_{\mathcal{N}(v)})_i - (\mathbf{h}_{\mathcal{N}(v)})_i \right| = O\left(\frac{\sqrt{d \log(d/\delta)}}{\epsilon \sqrt{|\mathcal{N}(v)|}} \right) \quad (9)$$

The above proposition indicates that with the mean aggregator function (which can be extended to other Aggregate functions as well), the estimation error decreases with a rate proportional to the square root of the node's degree. Therefore, the higher number of neighbors, the lower the estimation error. But as mentioned earlier, the size of $\mathcal{N}(v)$ is usually small in real graphs, which hinders the Aggregate function from driving out the injected noise on its own.

Algorithm 2: KProp Layer

Input : Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$; input vector $\mathbf{x}_v, \forall v \in \mathcal{V}$; linear aggregator function Aggregate; step parameter $K \geq 0$;
Output : Embedding vector $\mathbf{h}_v, \forall v \in \mathcal{V}$

```
for all v ∈ V do in parallel
    h^0_{N(v)} = x_v
    for k = 1 to K do
        h^k_{N(v)} = Aggregate({h^{k-1}_{N(u)}, ∀u ∈ N(v) − {v}})
    end
    h_v = h^K_{N(v)}
end
return {h_v, ∀v ∈ V}
```

In a different context, prior works have shown that considering higher-order neighbors can help learn better node representations [2, 28, 36]. Inspired by these works, a potential solution to this issue is to expand the neighborhood of each node v by considering more nodes that are not necessarily adjacent to v but reside within an adjustable local neighborhood around v. To this end, we use an efficient convolution layer, described in Algorithm 2, that can effectively be used to address the small-size neighborhood issue. The idea is simple: we aggregate features of those nodes that are up to K steps away from v by simply invoking the Aggregate function K consecutive times, without any non-linear transformation in between. For simplicity, we call this algorithm KProp, as every node propagates its message to K hops further.

As illustrated in Figure 2, we prepend KProp as a denoising layer to the GNN. This approach has two advantages: first, it allows to use any GNN architecture with any Aggregate function for the backbone model, as KProp already uses a linear Aggregate that satisfies Corollary 3.5; and second, it enables us to expand the effective aggregation set size for every node by controlling the step parameter K. However, it is essential to note that we cannot arbitrarily increase the neighborhood size around a node, since aggregating messages from too distant nodes could lead to over-smoothing of output vectors [30]. Therefore, there is a trade-off between the KProp's denoising accuracy and the overall GNN's expressive power.

It is worth mentioning that in KProp, we perform aggregations over $\mathcal{N}(v) - \{v\}$, i.e., we do not include self-loops. While it has been shown that adding self-loops can improve accuracy in conventional GNNs [26], excluding self-connections works better when dealing with noisy features. As K grows, with self-loops, we account for the injected noise in the feature vector of each node in the v's neighborhood multiple times in the aggregation. Therefore, removing self-loops helps to reduce the total noise by discarding repetitive node features from the aggregation.

3.3 Learning with private labels

In this last part, we describe the method used to perturb and collect labels privately and introduce our training algorithm for learning

locally private GNNs using perturbed labels, called label denoising with propagation (Drop). Let $f(\mathbf{x}) = \arg\max_{\mathbf{y}} \hat{p}(\mathbf{y} \mid \mathbf{x})$ be the target node classifier, where $\hat{p}(\mathbf{y} \mid \mathbf{x}) = g(\mathbf{x}, \mathcal{G}; \mathbf{W})$ approximates the class-conditional probabilities $p(\mathbf{y} \mid \mathbf{x})$ and is modeled by a GNN $g(.)$ with the learnable weight matrix $\mathbf{W}$. The goal is to optimize $\mathbf{W}$ such that $\hat{p}(\mathbf{y} \mid \mathbf{x})$ becomes as close as possible to $p(\mathbf{y} \mid \mathbf{x})$. In the standard setting, this is usually done by minimizing the cross-entropy loss function between $\hat{p}(\mathbf{y} \mid \mathbf{x})$ and true label $\mathbf{y}$ over the set of labeled nodes $\mathcal{V}_{\mathcal{L}}$:

$$\ell\left(\mathbf{y}, \hat{p}(\mathbf{y} \mid \mathbf{x})\right) = -\sum_{v \in \mathcal{V}_{\mathcal{L}}} \mathbf{y}_v^T \log \hat{p}(\mathbf{y} \mid \mathbf{x}_v) \tag{10}$$

However, since the labels are considered private, each node $v \in \mathcal{V}_{\mathcal{L}}$ that participates in the training procedure has to perturb their label $\mathbf{y}_v$ using some LDP mechanism, and send the perturbed label $\mathbf{y}'_v$ to the server. Still, if we train the GNN using the perturbed labels by minimizing the cross-entropy loss between $\hat{p}(\mathbf{y} \mid \mathbf{x})$ and perturbed labels $\mathbf{y}'$, namely $\ell\left(\mathbf{y}', \hat{p}(\mathbf{y} \mid \mathbf{x})\right)$, the model completely overfits the noisy labels and generalizes poorly to unseen nodes. However, many real-world graphs, such as social networks, are homophilic [34], meaning that nodes with structural similarity tend to have similar labels [48]. We exploit this fact to estimate the frequency of the labels in a local neighborhood around any node v to obtain its estimated label $\tilde{\mathbf{y}}_v$. To this end, we can use any LDP frequency oracle, such as randomized response [22], Unary Encoding [52], or Local Hashing [52]. In this paper, we use randomized response for two reasons: first, the number of classes is usually small, and randomized response has been shown to work better than other mechanisms in low dimensions [52]; and second, it introduces a symmetric, class-independent noise to the labels by flipping them according to the following distribution, which we later exploit in our learning algorithm:

$$p(\mathbf{y}' \mid \mathbf{y}) = \begin{cases} \frac{e^{\epsilon}}{e^{\epsilon}+c-1}, & \text{if } \mathbf{y}' = \mathbf{y} \\ \frac{1}{e^{\epsilon}+c-1}, & \text{otherwise} \end{cases} \tag{11}$$

where $\mathbf{y}$ and $\mathbf{y}'$ are clean and perturbed labels, respectively, c is the number of classes, and ϵ is the privacy budget.

Similar to estimating the graph convolution with noisy features, we also face the problem of small-size neighborhood if we only rely on the first-order neighbors to estimate the label frequency. In order to expand the neighborhood around each node, we take the same approach as we did for features: we apply KProp on node labels, i.e., we set $\tilde{y}_v = \arg\max_{i \in [c]} h_i(\mathbf{y}'_v, K_y)$ for all $v \in \mathcal{V}_{\mathcal{L}}$, where $h(.)$ is the KProp function, K_y is the step parameter, and $[c] = \{1, \ldots, c\}$. With the mean aggregator function, at every iteration, KProp updates every node's label distribution by averaging its neighbors' label distribution. In this paper, however, we instead use the GCN aggregator function [26]:

$$\textsc{Aggregate}\left(\{\mathbf{y}'_u, \forall u \in \mathcal{N}(v)\}\right) = \sum_{u \in \mathcal{N}(v)} \frac{\mathbf{y}'_u}{\sqrt{|\mathcal{N}(u)| \cdot |\mathcal{N}(v)|}} \tag{12}$$

Using the GCN aggregator leads to a lower estimation error than the mean aggregator due to the difference in their normalization factors, which affects their estimation variance. Specifically, the normalization factor in the GCN aggregator is $\sqrt{|\mathcal{N}(u)| \cdot |\mathcal{N}(v)|}$, while for the mean, it is $|\mathcal{N}(v)| = \sqrt{|\mathcal{N}(v)| \cdot |\mathcal{N}(v)|}$. In other words, the GCN aggregator considers the square root of the degree of both the central node v and its neighbor u, whereas the mean aggregator considers only the square root of the central node v's degree twice. Since there are many more low-degree nodes in many real graphs than high-degree ones, using the mean aggregator results in a small normalization factor for most nodes, leading to a higher estimation variance. But as many of the low-degree nodes are linked to the high-degree ones, the GCN aggregator balances the normalization by considering the degree of both link endpoints. Consequently, the normalization for many low-degree nodes increases compared to the mean aggregator, yielding a lower estimation variance.

As the step parameter K_y gradually increases, the estimated label $\tilde{\mathbf{y}}$ becomes more similar to the clean label $\mathbf{y}$. Therefore, an initial idea for the training algorithm would be to learn the GNN using $\tilde{\mathbf{y}}$ instead of $\mathbf{y}'$ by minimizing the cross-entropy loss between $\hat{p}(\mathbf{y} \mid \mathbf{x})$ and $\tilde{\mathbf{y}}$, namely $\ell\left(\tilde{\mathbf{y}}, \hat{p}(\mathbf{y} \mid \mathbf{x})\right)$. However, this approach has two downsides. First, it causes the GNN to become a predictor for $\tilde{\mathbf{y}}$ and not $\mathbf{y}$. Although $\tilde{\mathbf{y}}$ tend to converge to $\mathbf{y}$ as K_y increases, the output of KProp also becomes increasingly smoother, until the excessive KProp aggregations lead to over-smoothing, after which $\tilde{\mathbf{y}}$ will begin to diverge from $\mathbf{y}$ and become noisy again, while we are still fitting $\tilde{\mathbf{y}}$. Second, we cannot know how far we should increase K_y to get the best accuracy and prevent over-smoothing without clean validation data. One way to validate the model with noisy labels is to calculate the accuracy of the target classifier $f(\mathbf{x})$ for predicting the estimated label $\tilde{\mathbf{y}}$. However, suppose the model overfits the over-smoothed labels. In that case, the corresponding validation $\tilde{\mathbf{y}}$'s also becomes over-smoothed and can be well predicted by the model, resulting in a high validation but low test accuracy.

To address the first issue, instead of minimizing $\ell\left(\tilde{\mathbf{y}}, \hat{p}(\mathbf{y} \mid \mathbf{x})\right)$, we propose to minimize $\ell\left(\tilde{\mathbf{y}}, \hat{p}(\tilde{\mathbf{y}} \mid \mathbf{x})\right)$, i.e., the cross-entropy loss between the estimated label $\tilde{\mathbf{y}}$ and its approximated probability $\hat{p}(\tilde{\mathbf{y}} \mid \mathbf{x})$, which can be obtained by applying the same procedure on $\hat{p}(\mathbf{y} \mid \mathbf{x})$ as we did on $\mathbf{y}$ to obtain $\tilde{\mathbf{y}}$. In the first place, we applied randomized response on $\mathbf{y}$ to obtain $\mathbf{y}'$, and then passed the result to the KProp layer to get $\tilde{\mathbf{y}}$. If we go through the same steps to obtain $\hat{p}(\tilde{\mathbf{y}} \mid \mathbf{x})$ and then minimize its cross-entropy loss with $\tilde{\mathbf{y}}$, we can keep $\hat{p}(\mathbf{y} \mid \mathbf{x})$ intact when KProp causes over-smoothing, and at the same time benefit from it's denoising capability. To this end, we first need to calculate $\hat{p}(\mathbf{y}' \mid \mathbf{x})$ from $\hat{p}(\mathbf{y} \mid \mathbf{x})$:

$$\hat{p}(\mathbf{y}' \mid \mathbf{x}) = \sum_{\mathbf{y}} p(\mathbf{y}' \mid \mathbf{y}) \cdot \hat{p}(\mathbf{y} \mid \mathbf{x}) \tag{13}$$

where $p(\mathbf{y}' \mid \mathbf{y})$ is directly obtained from (11). This step would be analogous to applying randomized response to $\mathbf{y}$ and getting $\mathbf{y}'$. Finally, similar to applying KProp on $\mathbf{y}'$ to get $\tilde{\mathbf{y}}$, we treat $\hat{p}(\mathbf{y}' \mid \mathbf{x})$ as soft labels and apply KProp with the same step parameter to approximate $\hat{p}(\tilde{\mathbf{y}} \mid \mathbf{x})$:

$$\hat{p}(\tilde{\mathbf{y}} \mid \mathbf{x}) = softmax\left(h\left(\hat{p}(\mathbf{y}' \mid \mathbf{x}), K_y\right)\right) \tag{14}$$

where the softmax is used to normalize the KProp's output as a valid probability distribution. Finally, we train the model by minimizing $\ell\left(\tilde{\mathbf{y}}, \hat{p}(\tilde{\mathbf{y}} \mid \mathbf{x})\right)$.

To address the validation issue, we must make sure that our validation procedure is not affected by the KProp step parameter K_y. Clearly, if we use $\tilde{\mathbf{y}}$ for validation, by changing K_y we are also modifying estimated labels $\tilde{\mathbf{y}}$, and thus we are basically validating

different models with different labels. Therefore, we should only validate the model using the noisy labels $\mathbf{y}'$. Here, we choose the cross-entropy loss between $\mathbf{y}'$ and $\hat{p}(\mathbf{y}' \mid \mathbf{x})$, namely $\ell(\mathbf{y}', \hat{p}(\mathbf{y}' \mid \mathbf{x}))$, as Patrini *et al.* [39] show that this loss function, which they call forward correction loss, is unbiased, meaning that under expected label noise, $\ell(\mathbf{y}', \hat{p}(\mathbf{y}' \mid \mathbf{x}))$ is equal to $\ell(\mathbf{y}, \hat{p}(\mathbf{y} \mid \mathbf{x}))$, i.e., the original loss computed on clean data. Therefore, we train the GNN with different hyper-parameters, including K_y, and pick the one achieving the lowest forward correction loss.

While this is in principle a reasonable idea, the forward correction loss on its own is not enough to prevent overfitting. That's because when K_y is small, the estimated label $\tilde{\mathbf{y}}$ is more similar to the noisy one $\mathbf{y}'$ than the clean label $\mathbf{y}$, and thus the model overfits to the noisy labels. In this case, the GNN becomes a predictor for $\mathbf{y}'$ rather than $\mathbf{y}$, yielding a small forward correction loss. This is an incorrect validation signal as it favors K_y to be close to zero. To overcome this issue and detect overfitting to label noise, our approach is to look instead at the accuracy of the target classifier $f(\mathbf{x}) = \arg\max_{\mathbf{y}} \hat{p}(\mathbf{y} \mid \mathbf{x})$ for predicting the noisy labels $\mathbf{y}'$. From the randomized response algorithm, we know that the probability of keeping the label is $\frac{e^{\epsilon}}{e^{\epsilon}+c-1}$. This gives us an upper bound on the expected accuracy of a perfect classifier, i.e., the classifier with 100% accuracy on predicting clean labels. In other words, a perfect classifier can predict $\mathbf{y}'$ with an expected accuracy of at most $Acc^* = \frac{e^{\epsilon}}{e^{\epsilon}+c-1}$. Therefore, if during training the model we get accuracy above Acc^*, either on the training or validation dataset, we can consider it a signal of overfitting to the noisy labels. More specifically, we train the GNN for a maximum of T epochs and record the forward correction loss and the accuracy of the target classifier for predicting noisy labels over both training and validation sets at every epoch. At the end of training, we pick the model achieving the lowest forward correction loss such that their accuracy is at most Acc^*.

Putting all together, the pseudo-code of the LPGNN training algorithm with Drop is presented in Algorithm 3, where we use two different privacy budgets ϵ_x and ϵ_y for feature and label perturbation, respectively. The following corollary entails from our algorithm:

Corollary 3.7. *Algorithm 3 satisfies $(\epsilon_x + \epsilon_y)$-local differential privacy for graph nodes.*

Corollary 3.7 shows that the entire training procedure is LDP due to the robustness of differential privacy to post-processing [15]. Furthermore, any prediction performed by the LPGNN is again subject to the post-processing theorem [15], and therefore, satisfies LDP for the nodes, as the LDP mechanism is applied to the private data only once.

Algorithm 3: Locally Private GNN Training with Drop

Input : Graph $\mathcal{G} = (\mathcal{V}_{\mathcal{L}} \cup \mathcal{V}_{\mathcal{U}}, \mathcal{E})$; GNN model $g(\mathbf{x}, \mathcal{G}; \mathbf{W})$; KProp layer $h(\mathbf{x}, \mathcal{G}; K)$; KProp step parameter for features $K_x \geq 0$; KProp step parameter for labels $K_y \geq 0$; privacy budget for feature perturbation $\epsilon_x > 0$; privacy budget for label perturbation $\epsilon_y > 0$; range parameters α and β; number of classes c; maximum number of epochs T; learning rate η;

Output : Trained GNN weights $\mathbf{W}$

Server-side:
$\mathcal{V} \leftarrow \mathcal{V}_{\mathcal{L}} \cup \mathcal{V}_{\mathcal{U}}$
Send ϵ_x, ϵ_y, α, and β to every node $v \in \mathcal{V}$.
Node-side:
Obtain a perturbed vector $\mathbf{x}^*$ by Algorithm 1.
if *current node is in* $\mathcal{V}_{\mathcal{L}}$ **then**
  Obtain a perturbed label $\mathbf{y}'$ by (11).
else
  $\mathbf{y}' \leftarrow \vec{\mathbf{0}}$
end
Send $(\mathbf{x}^*, \mathbf{y}')$ to the server.
Server-side:
Obtain $\mathbf{x}'_v$ using (4) for all $v \in \mathcal{V}$.
$\mathbf{h}_v \leftarrow h(\mathbf{x}'_v, \mathcal{G}; K_x)$ for all $v \in \mathcal{V}$.
$\tilde{\mathbf{y}}_v \leftarrow h(\mathbf{y}'_v, \mathcal{G}; K_y)$ for all $v \in \mathcal{V}_{\mathcal{L}}$.
Partition $\mathcal{V}_{\mathcal{L}}$ into train and validation sets ${\mathcal{V}_{\mathcal{L}}}^{tr}$ and ${\mathcal{V}_{\mathcal{L}}}^{val}$.
$Acc^* \leftarrow e^{\epsilon_y} / (e^{\epsilon_y} + c - 1)$
for $t \in \{1, \ldots, T\}$ **do**
  for all $v \in \mathcal{V}_{\mathcal{L}}$ **do in parallel**
    $\hat{p}(\mathbf{y} \mid \mathbf{x}_v) \leftarrow g(\mathbf{h}_v, \mathcal{G}; \mathbf{W})$
    Obtain $\hat{p}(\mathbf{y}' \mid \mathbf{x}_v)$ using (13)
    Obtain $\hat{p}(\tilde{\mathbf{y}} \mid \mathbf{x}_v)$ using (14)
  end
  $\mathbf{W}^{t+1} \leftarrow \mathbf{W}^t - \eta \nabla \sum_{v \in {\mathcal{V}_{\mathcal{L}}}^{tr}} \ell(\tilde{\mathbf{y}}_v, \hat{p}(\tilde{\mathbf{y}} \mid \mathbf{x}_v))$
  $\ell^t_{val} \leftarrow \sum_{v \in {\mathcal{V}_{\mathcal{L}}}^{val}} \ell(\mathbf{y}'_v, \hat{p}(\mathbf{y}' \mid \mathbf{x}_v))$
  $Acc^t_{val} \leftarrow \frac{1}{|{\mathcal{V}_{\mathcal{L}}}^{val}|} \sum_{v \in {\mathcal{V}_{\mathcal{L}}}^{val}} Accuracy(\hat{p}(\mathbf{y} \mid \mathbf{x}_v), \mathbf{y}'_v)$
  $Acc^t_{tr} \leftarrow \frac{1}{|{\mathcal{V}_{\mathcal{L}}}^{tr}|} \sum_{v \in {\mathcal{V}_{\mathcal{L}}}^{tr}} Accuracy(\hat{p}(\mathbf{y} \mid \mathbf{x}_v), \mathbf{y}'_v)$
end
$t \leftarrow \arg\min_t \ell^t_{val}$ such that $Acc^t_{tr} \leq Acc^*$ and $Acc^t_{val} \leq Acc^*$
return $\mathbf{W}^t$

4 EXPERIMENTS

We conduct extensive experiments to assess the privacy-utility performance of the proposed method for the node classification task and evaluate it under different parameter settings that can affect its effectiveness.

4.1 Experimental settings

Datasets. We used two different sets of publicly available real-world datasets: two citation networks, Cora and Pubmed [61], which have a lower average degree, and two social networks, Facebook [42], and LastFM [43] that have a higher average degree. The description of the datasets is as followed:

- *Cora and Pubmed [61]:* These are well-known citation network datasets, where each node represents a document and edges denote citation links. Each node has a bag-of-words feature vector and a label indicating its category.
- *Facebook [42]:* This dataset is a page-page graph of verified Facebook sites. Nodes are official Facebook pages, and edges correspond to mutual likes between them. Node features are extracted from the site descriptions, and the labels denote site category.

Table 1: Descriptive statistics of the used datasets

Dataset	#Classes	#Nodes	#Edges	#Features	Avg. Deg.
Cora	7	2,708	5,278	1,433	3.90
Pubmed	3	19,717	44,324	500	4.50
Facebook	4	22,470	170,912	4,714	15.21
LastFM	10	7,083	25,814	7,842	7.29

- *LastFM [43]:* This social network is collected from the music streaming service LastFM. Nodes denote users from Asian countries, and links correspond to friendships. The task is to predict the home country of a user given the artists liked by them. Since the original dataset was highly imbalanced, we limited the classes to the top-10 having the most samples.

Summary statistics of the datasets are provided in Table 1.

Experiment setup. For all the datasets, we randomly split nodes into training, validation, and test sets with 50/25/25% ratios, respectively. Without loss of generality, we normalized the node features of all the datasets between zero and one[1], so in all cases, we have $\alpha = 0$ and $\beta = 1$. LDP feature perturbation is applied to the features of all the training, validation, and test sets. However, label perturbation is only applied to the training and validation sets, and the test set's labels are left clean for performance testing. We tried three state-of-the-art GNN architectures, namely GCN [26], GAT [47], and GraphSAGE [18], as the backbone model for LPGNN, with GraphSAGE being the default model for ablation studies. All the GNN models have two graph convolution layers with a hidden dimension of size 16 and the SeLU activation function [27] followed by dropout, and the GAT model has four attention heads. For both feature and label KProps, we use GCN aggregator function. We optimized the hyper-parameters of LPGNN based on the validation loss of GraphSAGE using the Drop algorithm as described in Section 3 with the following strategy, and used the same values for other GNN models: First, we fix K_x and K_y to $(16, 8)$, $(16, 2)$, $(4, 2)$, and $(8, 2)$, on Cora, Pubmed, Facebook, and LastFM, respectively, and for every pair of privacy budgets (ϵ_x, ϵ_y) in $(1, 1)$, $(1, \infty)$, $(\infty, 1)$, and (∞, ∞), we perform a grid search to find the best choices for initial learning rate and weight decay both from $\{10^{-4}, 10^{-3}, 10^{-2}\}$ and dropout rate from $\{10^{-4}, 10^{-3}, 10^{-2}\}$. Second, we fix the best found hyper-parameters in the previous step and search for the best performing KProp step parameters K_x and K_y both within $\{0, 2, 4, 8, 16\}$ for all $\epsilon_x \in \{0.01, 0.1, 1, 2, 3, \infty\}$ and $\epsilon_y \in \{0.5, 1, 2, 3, \infty\}$. More specifically, for every ϵ_x (resp. ϵ_y) except ∞, we use the best learning rate, weight decay, and dropout rate found for $\epsilon_x = 1$ in the previous step (resp. $\epsilon_y = 1$) to search for the best KProp step parameters. All the models are trained using the Adam optimizer [25] over a maximum of 500 epochs, and the best model is picked for testing based on the validation loss. We measured the accuracy on the test set over 10 consecutive runs and report the average and 95% confidence interval calculated by bootstrapping with 1000 samples. Our implementation is available at https://github.com/sisaman/LPGNN.

[1]Note that this normalization step does not affect the privacy, as the range parameters (α, β) are known to both the server and users, so the server could ask users to normalize their data between 0 and 1 before applying the multi-bit encoder.

4.2 Experimental results

Analyzing the utility-privacy trade-off. We first evaluate how our privacy-preserving LPGNN method performs under varying feature and label privacy budgets. We changed the feature privacy budget ϵ_x in $\{0.01, 0.1, 1, 2, 3, \infty\}$ and the label privacy budget within $\{1, 2, 3, \infty\}$. The cases where $\epsilon_x = \infty$ or $\epsilon_y = \infty$, are provided for comparison with non-private baselines, where we did not apply the corresponding LDP mechanism (multi-bit for features and randomized response for labels) and directly used the clean (non-perturbed) values. We performed this experiment using GCN, GAT, and GraphSAGE as different backbone GNN models and reported the node-classification accuracy, as illustrated in Figure 3.

We can observe that all the three GNN models demonstrate robustness to the perturbations, especially on features, and perform comparably to the non-private baselines. For instance, on the Cora dataset, both GCN and GraphSAGE could get an accuracy of about 80% at $\epsilon_x = 0.1$ and $\epsilon_y = 2$, which is only 6% lower than the non-private ($\epsilon = \infty$) method. On the other three datasets, we can decrease ϵ_x to 0.01 and ϵ_y to 1, and still get less than 10% accuracy loss compared to the non-private baseline. We believe that this is a very promising result, especially for a locally private model perturbing hundreds of features with a low privacy loss. This result shows that different components of LPGNN, from multi-bit mechanism to KProp, and the Drop algorithm are fitting well together.

According to the results, the GAT model slightly falls behind GCN and GraphSAGE in terms of accuracy-privacy trade-off, especially at high-privacy regimes $\epsilon_x \leq 1$, which is mainly due to its stronger dependence on the node features. Unlike the other two models, GAT uses node features at each layer to learn attention coefficients first, which are then used to weight different neighbors in the neighborhood aggregation. This property of GAT justifies its sensitivity to the features, and thus it degrades more than the other two models when the features are highly noisy. On the contrary, a model like GCN only uses node features in the GCN aggregator function (Eq. 12) and thus can better tolerate the noisy features. GraphSAGE averages neighboring node features and then appends the self feature vector to the aggregation, and therefore it is not as dependent as GAT on the features. Nevertheless, GAT could also achieve comparable results for $\epsilon_x \geq 1$ on all the datasets.

Analyzing the multi-bit mechanism. In Table 2, we compared the performance of our multi-bit mechanism (denoted as MB) against 1-bit mechanism (1B), Laplace mechanism (LP), and Analytic Gaussian mechanism (AG) [5]. The 1-bit mechanism [11], is obtained by setting $m = d$ in Algorithm 1. The Laplace and Gaussian mechanisms are two classic mechanisms that respectively add a zero-mean Laplace and Gaussian noise to the data with a noise variance calibrated based on the privacy budget, and are widely used for both single value and multidimensional data perturbation. Note that the Gaussian mechanism satisfies a relaxed version of ϵ-LDP , namely (ϵ, δ)-LDP, which (loosely speaking) means that it satisfies ϵ-LDP with probability at least $1 - \delta$ for $\delta > 0$. Here, we use the Analytic Gaussian mechanism [5], the optimized version of the standard Gaussian mechanism, with $\delta = 1^{-10}$. As all these mechanisms are used for feature perturbation, we set the label privacy budget $\epsilon_y = \infty$ and only consider their performance under different $\epsilon_x \in \{0.01, 0.1, 1, 2\}$. According to the results, our

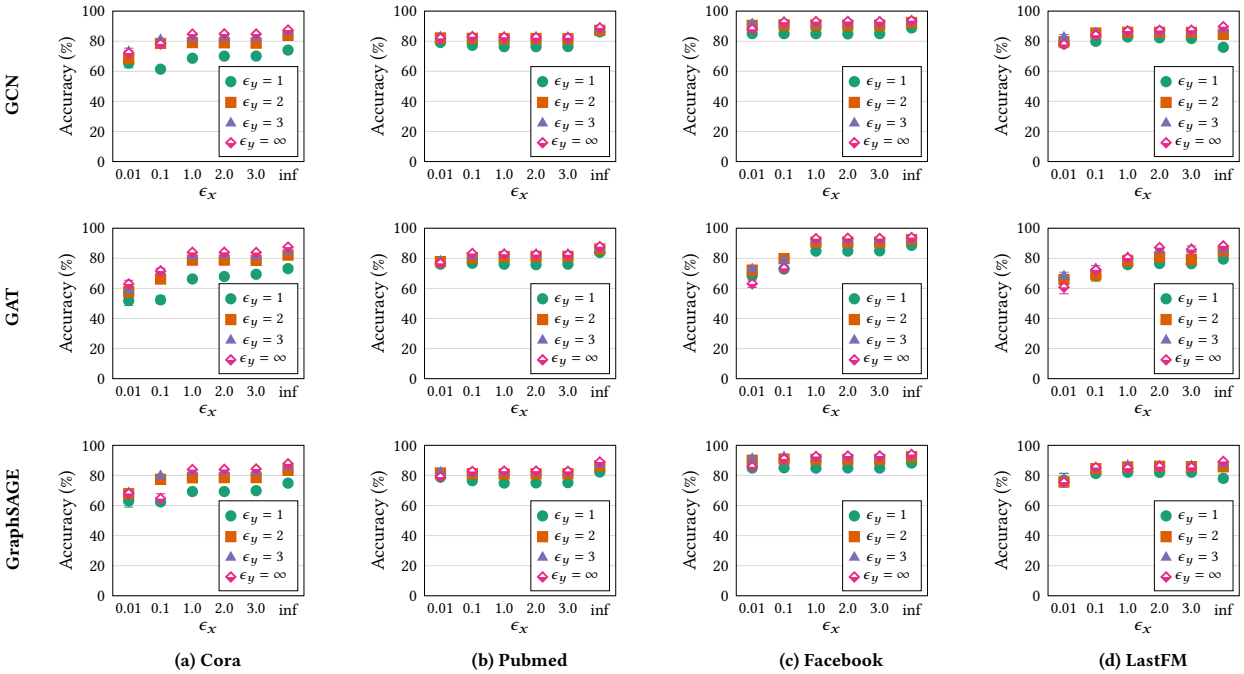


Figure 3: Comparison of LPGNN's performance with different GNN model under varying feature and label privacy budgets.

multi-bit mechanism consistently outperforms the other mechanisms in classification accuracy almost in all cases, especially under smaller privacy budgets. For instance, at $\epsilon_x = 0.01$, MB performs over 8%, 2%, 7%, and 12% better than the second-best mechanism AG on Cora, Pubmed, Facebook, and LastFM, respectively. This is mainly because the variance of our optimized multi-bit mechanism is lower than the other three, resulting in a more accurate estimation. Simultaneously, our mechanism is also efficient in terms of the communication overhead, requiring only two bits per feature. In contrast, the Gaussian mechanism's output is real-valued, usually taking 32 bits per feature (more or less, depending on the precision) to transmit a floating-point number.

To verify that using node features in a privacy-preserving manner has an added value in practice, in Table 3, we compare our multi-bit features with several ad-hoc feature vectors that can be used instead of the private features to train the GNN without any additional privacy cost. Ones is the all-one feature vector, Ohd is the one-hot encoding of the node's degree, as in [59], and Rnd is randomly initialized node features between 0 and 1. To have a fair comparison, we set the feature dimension of all the methods equal to the private features. We set $\epsilon_y = 1$ and compare the LPGNN's result with multi-bit encoded features under $\epsilon_x \in \{0.01, 0.1, 1\}$. We observe that LPGNN, with the multi-bit mechanism, even under the minimum privacy budget of 0.01, significantly outperforms the ad-hoc baselines in all cases, with an improvement ranging from around 7% on Facebook to over 20% on Pubmed comparing to the best performing ad-hoc baseline. Note that even though perturbed features under very small ϵ_x are noisier and becomes similar to Rnd, with the help of KProp, the resulting aggregation could estimate – even if poorly – the true aggregation, which might be enough for the GNN to distinguish between different neighborhoods. But in the case of Rnd, the aggregations carry no information about neighborhoods as the features are random, so the accuracy is worse than the multi-bit mechanism. This result shows that node features are also effective in addition to the graph structure, and we cannot ignore their utility.

Analyzing the effect of KProp. In this experiment, we investigate whether the KProp layer can effectively gain performance boost, for either node feature or labels. For this purpose, we varied the KProp's step parameters K_x and K_y both within $\{0, 2, 4, 8, 16\}$, and trained the LPGNN model under varying privacy budget, whose result is depicted in Figure 4. In the top row of the figure, we change $K_x \in \{0, 2, 4, 8, 16\}$ and $\epsilon_x \in \{0.01, 0.1, 1\}$, while fixing ϵ_y=1 and selecting the best values for K_y based on the validation loss. Conversely, in the bottom row, we vary $K_y \in \{0, 2, 4, 8, 16\}$ and $\epsilon_y \in \{0.5, 1, 2\}$, and set the best K_x at $\epsilon_x = 1$.

We observe that in all cases, both the feature and the label KProp layers are effective and can significantly boost the accuracy of the LPGNN depending on the dataset and the value of the corresponding privacy budget. Based on the results, the accuracy of LPGNN rises to an extent by increasing the step parameters, which shows that the model can benefit from larger population sizes to have a better estimation for both graph convolution and labels. Furthermore, we see that the maximum performance gain is different across the datasets and privacy budgets. As the estimates become more accurate due to an increase in the privacy budget, we see that KProp becomes less effective, mainly due to over-smoothing. But at

Table 2: Accuracy of LPGNN with different LDP mechanisms ($\epsilon_y = \infty$)

Dataset	Mech.	$\epsilon_x = 0.01$	$\epsilon_x = 0.1$	$\epsilon_x = 1$	$\epsilon_x = 2$
Cora	1B	45.8 ± 3.3	62.3 ± 1.5	59.9 ± 2.7	58.5 ± 2.9
	LP	43.2 ± 3.1	57.8 ± 2.3	61.9 ± 3.1	58.1 ± 2.1
	AG	59.7 ± 2.3	62.7 ± 2.8	67.5 ± 3.0	77.2 ± 1.9
	MB	**68.0 ± 2.9**	**64.6 ± 3.2**	**83.9 ± 0.4**	**84.0 ± 0.3**
Pubmed	1B	76.2 ± 0.6	74.8 ± 0.7	81.8 ± 0.4	82.5 ± 0.2
	LP	76.6 ± 0.5	75.2 ± 1.0	81.9 ± 0.4	82.4 ± 0.2
	AG	76.4 ± 0.6	81.5 ± 0.3	82.9 ± 0.2	**83.1 ± 0.2**
	MB	**78.9 ± 0.7**	**82.7 ± 0.2**	**82.9 ± 0.2**	82.9 ± 0.1
Facebook	1B	57.0 ± 3.4	76.3 ± 1.6	86.1 ± 0.6	84.0 ± 1.3
	LP	54.2 ± 2.9	72.5 ± 2.1	85.4 ± 0.4	84.8 ± 1.6
	AG	78.2 ± 1.4	85.6 ± 0.7	92.0 ± 0.1	92.4 ± 0.2
	MB	**85.8 ± 0.4**	**91.0 ± 0.4**	**92.7 ± 0.1**	**92.9 ± 0.1**
LastFm	1B	40.5 ± 7.4	56.2 ± 2.1	75.5 ± 2.5	68.1 ± 4.1
	LP	43.4 ± 5.7	50.5 ± 2.7	73.1 ± 2.9	67.2 ± 6.7
	AG	63.6 ± 2.4	75.1 ± 1.9	67.7 ± 4.2	63.5 ± 4.6
	MB	**75.6 ± 1.6**	**85.3 ± 0.4**	**84.9 ± 0.8**	**85.9 ± 1.1**

lower privacy budgets, KProp usually achieves the highest relative accuracy gain.

In the case of feature KProp, the performance is also correlated to the average node degree. For instance, at $\epsilon_x = 0.01$, on the social network datasets with a higher average degree, the accuracy gain is around 6% and 10% on Facebook and LastFM, respectively, while on lower-degree citation networks, it is over 20% on both Cora and Pubmed, which suggests that lower-degree datasets can benefit more from KProp. Furthermore, the optimal step parameter K_x that yields the best result also depends on the average degree of the graph. For example, we see that the trend is more or less increasing until the end for citation networks with a lower average degree. In contrast, the accuracy begins to fall over higher-degree social networks after $K_x = 4$. This means that in lower-degree datasets, KProp requires more steps to reach the sufficient number of nodes for aggregation, while on higher-degree graphs, it can achieve this number in fewer steps.

Regarding the label KProp, the performance growth depends not only on the average degree, but also on the number of classes, which can significantly affect the accuracy of randomized response. For instance, despite its high average degree, KProp could increase the accuracy on LastFM with 10 classes by over 20% at $\epsilon_y = 0.5$, while on the other high-degree dataset, Facebook, which has 4 classes, this number is at most 5%. Low-degree datasets still can benefit much from label KProp, with both Cora and Pubmed achieving a maximum of 30% accuracy boost at $\epsilon_y = 1$ and $\epsilon_y = 0.5$, respectively.

Investigating the Drop algorithm. In this final experiment, we investigate how using the Drop algorithm can affect the performance of LPGNN under different feature privacy budgets ϵ_y ranging with $\{0.5, 1.0, 2.0\}$, fixing $\epsilon_x = 1$. We compare the result of Drop with the classic cross-entropy, where we directly train the GNN with noisy labels. We also compare with the forward correction method [39], described in Section 3. Note that since our method does not rely on any clean validation data and is tailored for GNNs, it is not directly comparable to other general methods for deep learning with noisy labels that do not have these two characteristics. Table 4 presents the accuracy of different learning algorithms for the three label privacy budgets. It is evident that our Drop algorithm substantially outperforms the other two methods and can remarkably increase the final accuracy compared to the baselines, especially at high-privacy regimes with severe label noise, and also on datasets like LastFM with a high number of classes. Specifically, at $\epsilon_y = 0.5$, using Drop improves the accuracy of LPGNN by over 24%, 31%, 6%, and 25% on Cora, Pubmed, Facebook, and LastFM, respectively, compared to the forward correction method. As ϵ_y increases to 2, the labels become less noisy, so the accuracy difference between Drop and the other baselines shrinks. Still, Drop can perform better or at least equally compared to the forward correction method. This result suggests that Drop can effectively utilize the information within the graph structure to recover the actual node labels, and more importantly, it can achieve high accuracy without using any clean labels for model validation, e.g., for early stopping or hyper-parameter optimization.

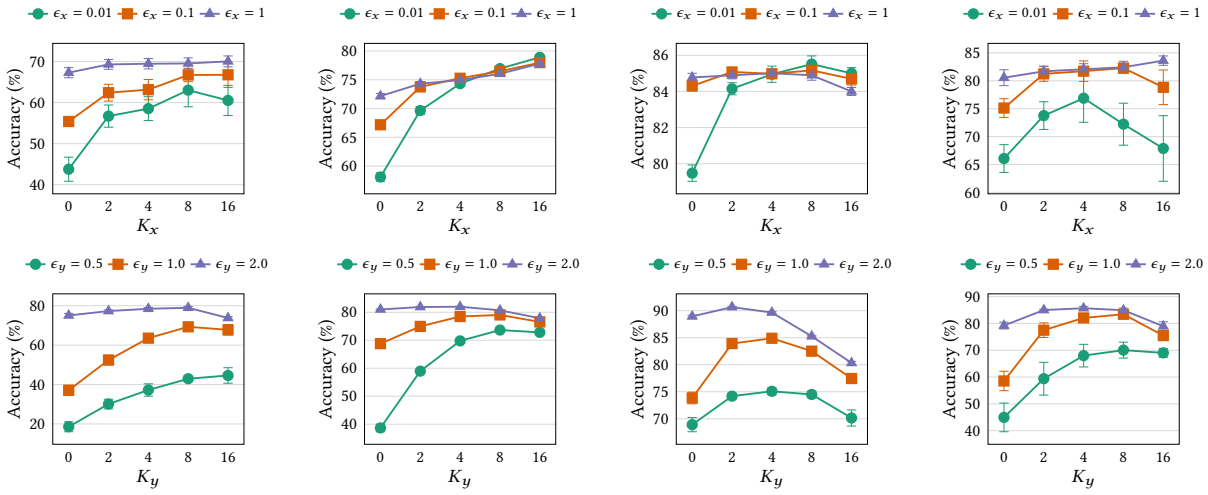
Table 3: Accuracy of LPGNN with different features ($\epsilon_y = 1$)

Feature	Cora	Pubmed	Facebook	LastFm
Ones	22.6 ± 5.0	38.9 ± 0.4	29.0 ± 1.4	19.6 ± 1.8
Ohd	44.4 ± 3.5	52.5 ± 5.7	77.2 ± 0.3	66.4 ± 1.6
Rnd	26.4 ± 3.0	56.0 ± 1.3	35.2 ± 5.6	32.3 ± 6.3
Mbm ($\epsilon_x = 0.01$)	63.0 ± 4.1	78.9 ± 0.2	85.0 ± 0.4	76.9 ± 4.3
Mbm ($\epsilon_x = 0.1$)	62.4 ± 2.0	76.5 ± 0.4	85.1 ± 0.2	81.2 ± 1.3
Mbm ($\epsilon_x = 1$)	69.3 ± 1.2	74.9 ± 0.3	84.9 ± 0.2	82.1 ± 1.0

5 RELATED WORK

Graph neural networks. Recent years have seen a surge in applying GNNs for representation learning over graphs, and numerous GNN models have been proposed for graph representation learning, including Graph Convolutional Networks [26], Graph Attention Networks [47], GraphSAGE [18], Graph Isomorphism Networks [59], Jumping Knowledge Networks [60], Gated Graph Neural Networks [32], and so on. We refer the reader to the available surveys on GNNs [19, 57] for other models and discussion on their performance and applications.

Local differential privacy. Local differential privacy has become increasingly popular for privacy-preserving data collection and analytics, as it does not need any trusted aggregator. There have been several LDP mechanisms on estimating aggregate statistics such as frequency [7, 16, 52], mean [11, 12, 50] heavy hitter [54], and frequent itemset mining [40]. There are also some works focusing on learning problems, such as probability distribution estimation



(a) Cora (b) Pubmed (c) Facebook (d) LastFM

Figure 4: Effect of the KProp step parameter on the performance of LPGNN. The top row depicts the effect of feature KProp with $\epsilon_y = 1$. The bottom row shows the effect of label KProp with $\epsilon_x = 1$. The y-axis is not set to zero to focus on the trends.

Table 4: Effect of Drop on the accuracy of LPGNN ($\epsilon_x = 1$)

Dataset	ϵ_y	Cross Entropy	Forward Correction	Drop
Cora	0.5	18.6 ± 1.3	18.6 ± 2.5	**42.9 ± 1.5**
	1.0	25.5 ± 1.7	37.1 ± 2.5	**69.3 ± 1.2**
	2.0	52.9 ± 2.1	75.1 ± 1.0	**78.4 ± 0.7**
Pubmed	0.5	37.1 ± 0.9	38.7 ± 1.4	**69.8 ± 0.7**
	1.0	65.4 ± 0.6	68.8 ± 0.7	**74.9 ± 0.3**
	2.0	80.5 ± 0.2	81.0 ± 0.2	**81.0 ± 0.2**
Facebook	0.5	50.9 ± 4.2	68.9 ± 1.3	**75.1 ± 0.6**
	1.0	55.2 ± 1.3	73.8 ± 1.1	**84.9 ± 0.2**
	2.0	81.6 ± 1.2	88.9 ± 0.2	**90.7 ± 0.1**
LastFm	0.5	21.1 ± 4.6	44.9 ± 5.3	**70.0 ± 3.0**
	1.0	28.4 ± 2.5	58.5 ± 3.6	**82.1 ± 1.0**
	2.0	56.8 ± 2.8	79.2 ± 1.3	**85.7 ± 0.7**

[3, 12, 22], heavy hitter discovery [6, 8, 54], frequent new term discovery [49], marginal release [10], clustering [37], and hypothesis testing [17]. Specifically, LDP frequency oracles are considered as fundamental primitives in LDP, and numerous mechanisms have been proposed [4, 6, 7, 16, 52, 62]. Most works rely on techniques like Hadamard transform [4, 6] and hashing [52]. LDP frequency oracles are also used in other tasks, e.g., frequent itemset mining [40, 53], and histogram estimation [22, 51, 55].

Privacy attacks on GNNs. Several recent works have attempted to characterize potential privacy attacks associated with GNNs and quantify the privacy leakage of publicly released GNN models or node embeddings that have been trained on private graph data. He *et al.* [1] proposed a series of link stealing attacks on a GNN model, to which the adversary has black-box access. They show that an adversary can accurately infer a link between any pair of nodes in a graph used to train the GNN model. Duddu *et al.* [13] presents a comprehensive study on quantifying the privacy leakage of graph embedding algorithms trained on sensitive graph data. More specifically, they introduce three major classes of privacy attacks on GNNs, namely membership inference, graph reconstruction, and attribute inference attack, under practical threat models and adversary assumptions. Finally, Wu *et al.* [56] propose a model extraction attack against GNNs by generating legitimate-looking queries as the normal nodes among the target graph, and then utilizing the query responses accessible structure knowledge to reconstruct the model. Overall, these works underline many privacy risks associated with GNNs and demonstrate the vulnerability of these models to various privacy attacks.

Privacy-preserving GNN models. While there is a growing interest in both theory and applications of GNNs, there have been relatively few attempts to provide privacy-preserving graph representation learning algorithms. Xu *et al.* [58] proposed a differentially private graph embedding method by applying the objective perturbation on the loss function of matrix factorization. Zhang and Ni [66] proposed a differentially private perturbed gradient descent method based on Lipschitz condition [20] for matrix factorization-based graph embedding. Both of these methods target classic graph embedding algorithms and not GNNs. Li *et al.* [29] presented a graph adversarial training framework that integrates disentangling and purging mechanisms to remove users' private information from learned node representations. Liao *et al.* [33] also follow an adversarial learning approach to address the attribute inference attack on GNNs, where they introduce a minimax game between the desired graph feature encoder and the worst-case attacker. However, both

of these works assume that the server has complete access to the private data, which is as opposed to our problem setting.

There are also recent approaches that attempted to address privacy in GNNs using federated and split learning. Mei *et al.* [35] proposed a GNN based on structural similarity and federated learning to hide content and structure information. Zhou *et al.* [68] tackled the problem of privacy-preserving node classification by splitting the computation graph of a GNN between multiple data holders and use a trusted server to combine the information from different parties and complete the training. However, as opposed to our method, these approaches rely on a trusted third party for model aggregation, and their privacy protection is not formally guaranteed. Finally, Jiang *et al.* [21] proposed a distributed and secure framework to learn the object representations in video data from graph sequences based on GNN and federated learning, and design secure aggregation primitives to protect privacy in federated learning. However, they assume that each party owns a series of graphs (extracted from video data), and the server uses federated learning to learn an inductive GNN over this distributed dataset of graphs, which is a different problem setting than the node data privacy we studied.

6 CONCLUSION

In this paper, we presented a locally private GNN to address node data privacy, where graph nodes have sensitive data that are kept private, but a central server could leverage them to train a GNN for learning rich node representations. To this end, we first proposed the *multi-bit mechanism*, a multidimensional ϵ-LDP algorithm that allows the server to privately collect node features and estimate the first-layer graph convolution of the GNN using the noisy features. Then, to further decrease the estimation error, we introduced KProp, a simple graph convolution layer that aggregates features from higher-order neighbors, which is prepended to the backbone GNN. Finally, to learn the model with perturbed labels, we proposed a learning algorithm called Drop that utilizes KProp for label denoising. Experimental results over real-world graph datasets on node classification demonstrated that the proposed framework could maintain an appropriate privacy-utility trade-off.

The concept of privacy-preserving graph representation learning is a novel field with many potential future directions that can go beyond node data privacy, such as link privacy and graph-level privacy. For the presented work, several future trends and improvements are imaginable. Firstly, in this paper, we protected the privacy of node features and labels, but the graph topology is left unprotected. Therefore, an important future work is to extend the current setting to preserving the graph structure as well. Secondly, we would like to explore other neighborhood expansion mechanisms that are more effective than the proposed KProp. Another future direction is to develop more rigorous algorithms for learning with differentially private labels, which is left unexplored for the case of GNNs. Finally, an interesting future work would be to combine the proposed LPGNN with deep graph learning algorithms to address privacy-preserving classification over non-relational datasets with low communication cost.

ACKNOWLEDGMENTS

This work was supported by the Swiss National Science Foundation (SNSF) through the Dusk2Dawn project (Sinergia program) under grant number 173696. Additional support was provided through the AI4Media project, funded by the European Commission (Grant 951911) under the H2020 Programme ICT-48-2020. We would like to thank Emiliano De Cristofaro, Hamed Haddadi, Nikolaos Karalias, and Mohammad Malekzadeh for their helpful comments on eariler drafts of this paper.

REFERENCES

[1] 2021. Stealing Links from Graph Neural Networks. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Vancouver, B.C. https://www.usenix.org/conference/usenixsecurity21/presentation/he

[2] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. 2019. M ix H op: Higher-Order Graph Convolutional Architectures via Sparsified Neighborhood Mixing *(Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, Long Beach, California, USA, 21–29.

[3] Jayadev Acharya, Ziteng Sun, and Huanyu Zhang. 2018. Communication efficient, sample optimal, linear time locally private discrete distribution estimation. *arXiv preprint arXiv:1802.04705* (2018).

[4] Jayadev Acharya, Ziteng Sun, and Huanyu Zhang. 2019. Hadamard response: Estimating distributions privately, efficiently, and with little communication. In *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, 1120–1129.

[5] Borja Balle and Yu-Xiang Wang. 2018. Improving the Gaussian Mechanism for Differential Privacy: Analytical Calibration and Optimal Denoising. In *International Conference on Machine Learning*. 394–403.

[6] Raef Bassily, Kobbi Nissim, Uri Stemmer, and Abhradeep Thakurta. 2017. Practical locally private heavy hitters. *arXiv preprint arXiv:1707.04982* (2017).

[7] Raef Bassily and Adam Smith. 2015. Local, private, efficient protocols for succinct histograms. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*. 127–135.

[8] Mark Bun, Jelani Nelson, and Uri Stemmer. 2019. Heavy hitters and the structure of local privacy. *ACM Transactions on Algorithms (TALG)* 15, 4 (2019), 1–40.

[9] Zhengdao Chen, Xiang Li, and Joan Bruna. 2017. Supervised community detection with line graph neural networks. *arXiv preprint arXiv:1705.08415* (2017).

[10] Graham Cormode, Tejas Kulkarni, and Divesh Srivastava. 2018. Marginal release under local differential privacy. In *Proceedings of the 2018 International Conference on Management of Data*. 131–146.

[11] Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. 2017. Collecting telemetry data privately. In *Advances in Neural Information Processing Systems*. 3571–3580.

[12] John C Duchi, Michael I Jordan, and Martin J Wainwright. 2018. Minimax optimal procedures for locally private estimation. *J. Amer. Statist. Assoc.* 113, 521 (2018), 182–201.

[13] Vasisht Duddu, Antoine Boutet, and Virat Shejwalkar. 2020. Quantifying Privacy Leakage in Graph Embedding. In *Mobiquitous 2020-17th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*. 1–11.

[14] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Al á n Aspuru-Guzik, and Ryan P Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*. 2224–2232.

[15] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends ® in Theoretical Computer Science* 9, 3–4 (2014), 211–407.

[16] Ú lfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. 2014. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*. 1054–1067.

[17] Marco Gaboardi and Ryan Rogers. 2018. Local private hypothesis testing: Chi-square tests. In *International Conference on Machine Learning*. PMLR, 1626–1635.

[18] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in neural information processing systems*. 1024–1034.

[19] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584* (2017).

[20] Madhav Jha and Sofya Raskhodnikova. 2013. Testing and reconstruction of Lipschitz functions with applications to data privacy. *SIAM J. Comput.* 42, 2 (2013), 700–731.

[21] Meng Jiang, Taeho Jung, Ryan Karl, and Tong Zhao. 2020. Federated Dynamic GNN with Secure Aggregation. *arXiv preprint arXiv:2009.07351* (2020).

[22] Peter Kairouz, Keith Bonawitz, and Daniel Ramage. 2016. Discrete distribution estimation under local privacy. In *International Conference on Machine Learning*. PMLR, 2436–2444.

[23] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. 2019. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977* (2019).

[24] Shiva Prasad Kasiviswanathan, Homin K Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. 2011. What can we learn privately? *SIAM J. Comput.* 40, 3 (2011), 793–826.

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

[26] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR)*.

[27] G ü nter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. 2017. Self-normalizing neural networks. In *Advances in neural information processing systems*. 971–980.

[28] Johannes Klicpera, Stefan Wei ß enberger, and Stephan G ü nnemann. 2019. Diffusion improves graph learning. In *Advances in Neural Information Processing Systems*. 13354–13366.

[29] Kaiyang Li, Guangchun Luo, Yang Ye, Wei Li, Shihao Ji, and Zhipeng Cai. 2020. Adversarial Privacy Preserving Graph Embedding against Inference Attack. *arXiv preprint arXiv:2008.13072* (2020).

[30] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. *arXiv preprint arXiv:1801.07606* (2018).

[31] Yayong Li, Ling Chen, et al. 2021. Unified Robust Training for Graph NeuralNetworks against Label Noise. *arXiv preprint arXiv:2103.03414* (2021).

[32] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2015. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493* (2015).

[33] Peiyuan Liao, Han Zhao, Keyulu Xu, Tommi Jaakkola, Geoffrey Gordon, Stefanie Jegelka, and Ruslan Salakhutdinov. 2020. Graph Adversarial Networks: Protecting Information against Adversarial Attacks. *arXiv preprint arXiv:2009.13504* (2020).

[34] Miller McPherson, Lynn Smith-Lovin, and James M Cook. 2001. Birds of a feather: Homophily in social networks. *Annual review of sociology* 27, 1 (2001), 415–444.

[35] G. Mei, Z. Guo, S. Liu, and L. Pan. 2019. SGNN: A Graph Neural Network Based Federated Learning Approach by Hiding Structure. In *2019 IEEE International Conference on Big Data (Big Data)*. IEEE Computer Society, Los Alamitos, CA, USA, 2560–2568.

[36] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. 2019. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4602–4609.

[37] Kobbi Nissim and Uri Stemmer. 2018. Clustering algorithms for the centralized and local models. In *Algorithmic Learning Theory*. PMLR, 619–653.

[38] Hoang NT, Choong Jun Jin, and Tsuyoshi Murata. 2019. Learning graph neural networks with noisy labels. *arXiv preprint arXiv:1905.01591* (2019).

[39] Giorgio Patrini, Alessandro Rozza, Aditya Krishna Menon, Richard Nock, and Lizhen Qu. 2017. Making deep neural networks robust to label noise: A loss correction approach. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1944–1952.

[40] Zhan Qin, Yin Yang, Ting Yu, Issa Khalil, Xiaokui Xiao, and Kui Ren. 2016. Heavy hitter estimation over set-valued data with local differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 192–203.

[41] Sungmin Rhee, Seokjun Seo, and Sun Kim. 2017. Hybrid approach of relation network and localized graph convolutional filtering for breast cancer subtype classification. *arXiv preprint arXiv:1711.05859* (2017).

[42] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. 2019. Multi-scale Attributed Node Embedding. *arXiv preprint arXiv:1909.13021* (2019).

[43] Benedek Rozemberczki and Rik Sarkar. 2020. Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20)*. ACM.

[44] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE Transactions on Neural Networks* 20, 1 (2008), 61–80.

[45] Hwanjun Song, Minseok Kim, Dongmin Park, and Jae-Gil Lee. 2020. Learning from noisy labels with deep neural networks: A survey. *arXiv preprint arXiv:2007.08199* (2020).

[46] Abhradeep Guha Thakurta, Andrew H Vyrros, Umesh S Vaishampayan, Gaurav Kapoor, Julien Freudiger, Vivek Rangarajan Sridhar, and Doug Davidson. 2017. Learning new words. US Patent 9,594,741.

[47] Petar Veli č kovi ć, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[48] Hongwei Wang and Jure Leskovec. 2020. Unifying graph convolutional neural networks and label propagation. *arXiv preprint arXiv:2002.06755* (2020).

[49] Ning Wang, Xiaokui Xiao, Yin Yang, Ta Duy Hoang, Hyejin Shin, Junbum Shin, and Ge Yu. 2018. PrivTrie: Effective frequent term discovery under local differential privacy. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 821–832.

[50] Ning Wang, Xiaokui Xiao, Yin Yang, Jun Zhao, Siu Cheung Hui, Hyejin Shin, Junbum Shin, and Ge Yu. 2019. Collecting and analyzing multidimensional data with local differential privacy. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 638–649.

[51] Shaowei Wang, Liusheng Huang, Pengzhan Wang, Yiwen Nie, Hongli Xu, Wei Yang, Xiang-Yang Li, and Chunming Qiao. 2016. Mutual information optimally local private discrete distribution estimation. *arXiv preprint arXiv:1607.08025* (2016).

[52] Tianhao Wang, Jeremiah Blocki, Ninghui Li, and Somesh Jha. 2017. Locally differentially private protocols for frequency estimation. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*. 729–745.

[53] Tianhao Wang, Ninghui Li, and Somesh Jha. 2018. Locally differentially private frequent itemset mining. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 127–143.

[54] Tianhao Wang, Ninghui Li, and Somesh Jha. 2019. Locally differentially private heavy hitter identification. *IEEE Transactions on Dependable and Secure Computing* (2019).

[55] Yue Wang, Xintao Wu, and Donghui Hu. 2016. Using Randomized Response for Differential Privacy Preserving Data Collection.. In *EDBT/ICDT Workshops*, Vol. 1558. 0090–6778.

[56] Bang Wu, Xiangwen Yang, Shirui Pan, and Xingliang Yuan. 2020. Model Extraction Attacks on Graph Neural Networks: Taxonomy and Realization. *arXiv preprint arXiv:2010.12751* (2020).

[57] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[58] Depeng Xu, Shuhan Yuan, Xintao Wu, and HaiNhat Phan. 2018. DPNE: Differentially private network embedding. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 235–246.

[59] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).

[60] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer Dy and Andreas Krause (Eds.). PMLR, Stockholmsmässan, Stockholm Sweden, 5453–5462.

[61] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. 2016. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861* (2016).

[62] Min Ye and Alexander Barg. 2018. Optimal schemes for discrete distribution estimation under locally differential privacy. *IEEE Transactions on Information Theory* 64, 8 (2018), 5662–5676.

[63] Kun Yi and Jianxin Wu. 2019. Probabilistic end-to-end noise correction for learning with noisy labels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 7017–7025.

[64] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. 2017. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412* (2017).

[65] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*. 5165–5175.

[66] Sen Zhang and Weiwei Ni. 2019. Graph Embedding Matrix Sharing With Differential Privacy. *IEEE Access* 7 (2019), 89390–89399.

[67] Zhilu Zhang and Mert R Sabuncu. 2018. Generalized cross entropy loss for training deep neural networks with noisy labels. *arXiv preprint arXiv:1805.07836* (2018).

[68] Jun Zhou, Chaochao Chen, Longfei Zheng, Xiaolin Zheng, Bingzhe Wu, Ziqi Liu, and Li Wang. 2020. Privacy-Preserving Graph Neural Network for Node Classification. *arXiv preprint arXiv:2005.11903* (2020).

A DEFERRED THEORETICAL ARGUMENTS

A.1 Theorem 3.1

Proof. Let $\mathcal{M}(\mathbf{x})$ denote the multi-bit encoder (Algorithm 1) applied on the input vector $\mathbf{x}$. Let $\mathbf{x}^* = \mathcal{M}(\mathbf{x})$ be the encoded vector corresponding to $\mathbf{x}$. We need to show that for any two input features $\mathbf{x}_1$ and $\mathbf{x}_2$, we have $\frac{\Pr[\mathcal{M}(\mathbf{x}_1)=\mathbf{x}^*]}{\Pr[\mathcal{M}(\mathbf{x}_2)=\mathbf{x}^*]} \leq e^{\epsilon}$.

According to Algorithm 1, for any dimension $i \in \{1, 2, \ldots, d\}$, it can be easily seen that $x_i^* \in \{-1, 0, 1\}$. The case $x_i^* = 0$ occurs when $i \notin \mathcal{S}$ with probability $1 - \frac{m}{d}$, therefore:

$$\frac{\Pr\left[\mathcal{M}(\mathbf{x}_1)_i = 0\right]}{\Pr\left[\mathcal{M}(\mathbf{x}_2)_i = 0\right]} = \frac{1 - m/d}{1 - m/d} = 1 \leq e^{\epsilon}, \quad \forall \epsilon > 0 \tag{15}$$

According to Algorithm 1, in the case of $x_i^* \in \{-1, 1\}$, we see that the probability of getting $x_i^* = 1$ ranges from $\frac{m}{d} \cdot \frac{1}{e^{\epsilon/m}+1}$ to $\frac{m}{d} \cdot \frac{e^{\epsilon/m}}{e^{\epsilon/m}+1}$ depending on the value of x_i. Analogously, the probability of $x_i^* = -1$ also varies from $\frac{m}{d} \cdot \frac{1}{e^{\epsilon/m}+1}$ to $\frac{m}{d} \cdot \frac{e^{\epsilon/m}}{e^{\epsilon/m}+1}$. Therefore:

$$\begin{aligned}\frac{\Pr\left[\mathcal{M}(\mathbf{x}_1)_i \in \{-1, 1\}\right]}{\Pr\left[\mathcal{M}(\mathbf{x}_2)_i \in \{-1, 1\}\right]} &\leq \frac{\max \Pr\left[\mathcal{M}(\mathbf{x}_1)_i \in \{-1, 1\}\right]}{\min \Pr\left[\mathcal{M}(\mathbf{x}_2)_i \in \{-1, 1\}\right]} \\ &\leq \frac{\frac{m}{d} \cdot \frac{e^{\epsilon/m}}{e^{\epsilon/m}+1}}{\frac{m}{d} \cdot \frac{1}{e^{\epsilon/m}+1}} \leq e^{\epsilon/m}\end{aligned} \tag{16}$$

Consequently, we have:

$$\begin{aligned}\frac{\Pr\left[\mathcal{M}(\mathbf{x}_1) = \mathbf{x}^*\right]}{\Pr\left[\mathcal{M}(\mathbf{x}_2) = \mathbf{x}^*\right]} &= \prod_{i=1}^{d} \frac{\Pr\left[\mathcal{M}(x_1)_i = x_i^*\right]}{\Pr\left[\mathcal{M}(x_2)_i = x_i^*\right]} \\ &= \prod_{j \mid x_j^* = 0} \frac{\Pr\left[\mathcal{M}(x_1)_j = 0\right]}{\Pr\left[\mathcal{M}(x_2)_j = 0\right]} \\ &\quad \times \prod_{k \mid x_k^* \in \{-1,1\}} \frac{\Pr\left[\mathcal{M}(x_1)_k \in \{-1, 1\}\right]}{\Pr\left[\mathcal{M}(x_2)_k \in \{-1, 1\}\right]} \qquad (17) \\ &= \prod_{x_k^* \in \{-1,1\}} \frac{\Pr\left[\mathcal{M}(x_1)_k \in \{-1, 1\}\right]}{\Pr\left[\mathcal{M}(x_2)_k \in \{-1, 1\}\right]} \qquad (18) \\ &\leq \prod_{x_k^* \in \{-1,1\}} e^{\epsilon/m} \qquad (19) \\ &\leq e^{\epsilon} \qquad (20)\end{aligned}$$

which concludes the proof. In the above, (18) and (19) follows from applying (15) and (16), respectively, and (20) follows from the fact that exactly m number of input features result in non-zero output. □

A.2 Proposition 3.2

We first establish the following lemma and then prove Proposition 3.2:

Lemma A.1. *Let $\mathbf{x}^*$ be the output of Algorithm 1 on the input vector $\mathbf{x}$. For any dimension $i \in \{1, 2, \ldots, d\}$, we have:*

$$\mathbb{E}\left[x_i^*\right] = \frac{m}{d} \cdot \frac{e^{\epsilon/m} - 1}{e^{\epsilon/m} + 1} \cdot \left(2 \cdot \frac{x_i - \alpha}{\beta - \alpha} - 1\right) \tag{21}$$

and

$$Var\left[x_i^*\right] = \frac{m}{d} - \left[\frac{m}{d} \cdot \frac{e^{\epsilon/m} - 1}{e^{\epsilon/m} + 1} \cdot \left(2 \cdot \frac{x_i - \alpha}{\beta - \alpha} - 1\right)\right]^2 \tag{22}$$

Proof. For the expectation, we have:

$$\begin{aligned}\mathbb{E}\left[x_i^*\right] &= \mathbb{E}\left[x_i^* \mid s_i = 0\right] \Pr(s_i = 0) + \mathbb{E}\left[x_i^* \mid s_i = 1\right] \Pr(s_i = 1) \\ &= \frac{m}{d} \cdot (2\,\mathbb{E}\left[t_i\right] - 1)\end{aligned} \tag{23}$$

Since t_i is a Bernoulli random variable, we have:

$$\mathbb{E}\left[t_i\right] = \frac{1}{e^{\epsilon/m} + 1} + \frac{x_i - \alpha}{\beta - \alpha} \cdot \frac{e^{\epsilon/m} - 1}{e^{\epsilon/m} + 1} \tag{24}$$

Combining (23) and (24) yields:

$$\begin{aligned}\mathbb{E}\left[x_i^*\right] &= \frac{m}{d} \cdot \left[2\left(\frac{1}{e^{\epsilon/m} + 1} + \frac{x_i - \alpha}{\beta - \alpha} \cdot \frac{e^{\epsilon/m} - 1}{e^{\epsilon/m} + 1}\right) - 1\right] \\ &= \frac{m}{d} \cdot \left[\frac{1 - e^{\epsilon/m}}{e^{\epsilon/m} + 1} + 2 \cdot \frac{x_i - \alpha}{\beta - \alpha} \cdot \frac{e^{\epsilon/m} - 1}{e^{\epsilon/m} + 1}\right] \\ &= \frac{m}{d} \cdot \frac{e^{\epsilon/m} - 1}{e^{\epsilon/m} + 1} \cdot \left(2 \cdot \frac{x_i - \alpha}{\beta - \alpha} - 1\right)\end{aligned} \tag{25}$$

For the variance, we have:

$$\begin{aligned}Var\left[x_i^*\right] &= \mathbb{E}\left[\left(x_i^*\right)^2\right] - \mathbb{E}\left[x_i^*\right]^2 \\ &= \mathbb{E}\left[\left(x_i^*\right)^2 \mid s_i = 0\right] \Pr(s_i = 0) \\ &\quad + \mathbb{E}\left[\left(x_i^*\right)^2 \mid s_i = 1\right] \Pr(s_i = 1) - \mathbb{E}\left[x_i^*\right]^2\end{aligned}$$

Given $s_i = 1$, we have $x_i^* = \pm 1$, and thus $\left(x_i^*\right)^2 = 1$. Therefore, combining with (25), we get:

$$Var\left[x_i^*\right] = \frac{m}{d} - \left[\frac{m}{d} \cdot \frac{e^{\epsilon/m} - 1}{e^{\epsilon/m} + 1} \cdot \left(2 \cdot \frac{x_i - \alpha}{\beta - \alpha} - 1\right)\right]^2 \tag{26}$$

□

Now we prove Proposition 3.2.

Proof. We need to show that $\mathbb{E}\left[x'_{v,i}\right] = x_{v,i}$ for any $v \in \mathcal{V}$ and any dimension $i \in \{1, 2, \ldots, d\}$.

$$\mathbb{E}\left[x'_{v,i}\right] = \frac{d(\beta - \alpha)}{2m} \cdot \frac{e^{\epsilon/m} + 1}{e^{\epsilon/m} - 1} \cdot \mathbb{E}\left[x_{v,i}^*\right] + \frac{\alpha + \beta}{2} \tag{27}$$

Applying Lemma A.1 yields:

$$\begin{aligned}\mathbb{E}\left[x'_{v,i}\right] &= \frac{d(\beta - \alpha)}{2m} \frac{e^{\epsilon/m} + 1}{e^{\epsilon/m} - 1} \left[\frac{m}{d} \frac{e^{\epsilon/m} - 1}{e^{\epsilon/m} + 1} \left(2\frac{x_{v,i} - \alpha}{\beta - \alpha} - 1\right)\right] \\ &\quad + \frac{\alpha + \beta}{2} \\ &= \frac{\beta - \alpha}{2} \left(2\frac{x_{v,i} - \alpha}{\beta - \alpha} - 1\right) + \frac{\alpha + \beta}{2} \\ &= x_{v,i} - \alpha - \frac{\beta - \alpha}{2} + \frac{\alpha + \beta}{2} = x_{v,i}\end{aligned}$$

□

A.3 Proposition 3.3

Proof. According to (4), the variance of $x'_{v,i}$ can be written in terms of the variance of $x_{v,i}^*$ as:

$$Var\left[x'_{v,i}\right] = \left[\frac{d(\beta - \alpha)}{2m} \cdot \frac{e^{\epsilon/m} + 1}{e^{\epsilon/m} - 1}\right]^2 \cdot Var\left[x_{v,i}^*\right]$$

Applying Lemma A.1 yields:

$$
\begin{aligned}
Var\left[x'_{v,i}\right] &= \left[\frac{d(\beta-\alpha)}{2m}\cdot\frac{e^{\epsilon/m}+1}{e^{\epsilon/m}-1}\right]^2 \\
&\quad\times\left(\frac{m}{d}-\left[\frac{m}{d}\cdot\frac{e^{\epsilon/m}-1}{e^{\epsilon/m}+1}\cdot\left(2\cdot\frac{x_{v,i}-\alpha}{\beta-\alpha}-1\right)\right]^2\right) \\
&= \frac{d}{m}\cdot\left(\frac{\beta-\alpha}{2}\cdot\frac{e^{\epsilon/m}+1}{e^{\epsilon/m}-1}\right)^2 \\
&\quad-\left[\frac{\beta-\alpha}{2}\cdot\left(2\cdot\frac{x_{v,i}-\alpha}{\beta-\alpha}-1\right)\right]^2 \\
&= \frac{d}{m}\cdot\left(\frac{\beta-\alpha}{2}\cdot\frac{e^{\epsilon/m}+1}{e^{\epsilon/m}-1}\right)^2-\left(x_{v,i}-\frac{\alpha+\beta}{2}\right)^2
\end{aligned}
$$

□

A.4 Proposition 3.4

Proof. We look for a value of m that minimizes the variance of the multi-bit rectifier defined by (4), i.e., $Var[x'_{v,i}]$, for any arbitrary node $v \in \mathcal{V}$ and any arbitrary dimension $i \in \{1,,2,\ldots,d\}$. However, based on Proposition 3.3, $Var[x'_{v,i}]$ depends on the private feature $x_{v,i}$, which is unknown to the server. Therefore, we find the optimal m, denoted by $m^\star$, by minimizing the upperbound of the variance:

$$m^\star = \arg\min_m \max_x Var[x'] \tag{28}$$

where we omitted the node v and dimension i subscripts for simplicity. From Proposition 3.3, it can be easily seen that the variance is maximized when $x = \frac{\alpha+\beta}{2}$, which yields:

$$\max_x Var[x'] = \frac{d}{m}\cdot\left(\frac{\beta-\alpha}{2}\cdot\frac{e^{\epsilon/m}+1}{e^{\epsilon/m}-1}\right)^2 \tag{29}$$

$$= C\cdot z\cdot\left(\frac{e^z+1}{e^z-1}\right)^2 = C\cdot z\cdot\coth^2\left(\frac{z}{2}\right) \tag{30}$$

where we set $z = \frac{\epsilon}{m}$ and $C = \frac{d}{\epsilon}\cdot\left(\frac{\beta-\alpha}{2}\right)^2$, and coth(.) is the hyperbolic cotangent. Therefore, minimizing (29) with respect to m is equivalent to minimizing (30) with respect to z, and then recover $m^\star$ as $\frac{\epsilon}{z^\star}$, where $z^\star$ is the optimal z minimizing (30). More formally:

$$
\begin{aligned}
z^\star &= \arg\min_z\left[C\cdot z\cdot\coth^2\left(\frac{z}{2}\right)\right] \\
&= \arg\min_z\left[z\cdot\coth^2\left(\frac{z}{2}\right)\right]
\end{aligned}
$$

where the constant C were dropped as it does not depend on z. The function $f(z) = z\cdot\coth^2(\frac{z}{2})$ is a convex function with a single minimum on $(0,\infty)$, as shown in Figure 5. Taking the derivative of $f(.)$ with respect to z and set it to zero gives us the minimum:

$$f'(z) = \frac{d}{dz}z\cdot\coth^2\left(\frac{z}{2}\right) = \coth\left(\frac{z}{2}\right)\left[\coth\left(\frac{z}{2}\right)-z\cdot\operatorname{csch}^2\left(\frac{z}{2}\right)\right] = 0$$

and then we have:

$$z = \frac{\coth(\frac{z}{2})}{\operatorname{csch}^2(\frac{z}{2})} = \frac{\sinh(z)}{2} \tag{31}$$

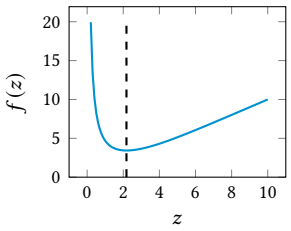


Figure 5: Plotting $f(z) = z\cdot\coth^2(\frac{z}{2})$. The gray dashed line indicate the location of the minimum.

Solving the above equation yields $z^\star \simeq 2.18$, and therefore we have $m^\star = \frac{\epsilon}{2.18}$. However, m should be an integer value between 1 and d. To enforce this, we set:

$$m^\star = \max(1, \min(d, \left\lfloor\frac{\epsilon}{2.18}\right\rfloor)) \tag{32}$$

□

A.5 Corollary 3.5

Proof. We need to show that the following holds for any node $v \in \mathcal{V}$:

$$\mathbb{E}\left[\widehat{\mathbf{h}}_{\mathcal{N}(v)}\right] = \mathbf{h}_{\mathcal{N}(v)}$$

The left hand side of the above can be written as:

$$\mathbb{E}\left[\widehat{\mathbf{h}}_{\mathcal{N}(v)}\right] = \mathbb{E}\left[\textsc{Aggregate}\left(\{\mathbf{x}'_u, \forall u\in\mathcal{N}(v)\}\right)\right]$$

Since Aggregate is linear, due to the linearity of expectation, the expectation sign can be moved inside Aggregate:

$$\mathbb{E}\left[\widehat{\mathbf{h}}_{\mathcal{N}(v)}\right] = \textsc{Aggregate}\left(\{\mathbb{E}\left[\mathbf{x}'_u\right], \forall u\in\mathcal{N}(v)\}\right)$$

Finally, by Proposition 3.2, we have:

$$\mathbb{E}\left[\widehat{\mathbf{h}}_{\mathcal{N}(v)}\right] = \textsc{Aggregate}\left(\{\mathbf{x}_u, \forall u\in\mathcal{N}(v)\}\right) = \mathbf{h}_{\mathcal{N}(v)}$$

□

A.6 Proposition 3.6

Proof. According to (4) and depending on Algorithm 1's output, for any node $u \in \mathcal{V}$ and any dimension $i \in \{1,2,\ldots,d\}$, we have:

$$x'_{u,i} = \begin{cases} \frac{\alpha+\beta}{2} - \frac{d(\beta-\alpha)}{2m}\cdot\frac{e^{\epsilon/m}+1}{e^{\epsilon/m}-1} & \text{if } x^*_{u,i} = -1 \\ \frac{\alpha+\beta}{2} & \text{if } x^*_{u,i} = 0 \\ \frac{\alpha+\beta}{2} + \frac{d(\beta-\alpha)}{2m}\cdot\frac{e^{\epsilon/m}+1}{e^{\epsilon/m}-1} & \text{if } x^*_{u,i} = 1 \end{cases}$$

and therefore

$$x'_{u,i} \in [\frac{\alpha+\beta}{2} - C, \frac{\alpha+\beta}{2} + C]$$

where

$$C = \frac{d(\beta-\alpha)}{2m}\cdot\frac{e^{\epsilon/m}+1}{e^{\epsilon/m}-1} \tag{33}$$

Therefore, considering that $x_{u,i} \in [\alpha,\beta]$, we get:

$$\left|x'_{u,i} - x_{u,i}\right| \le \frac{\beta-\alpha}{2} + C \tag{34}$$

and also by Proposition 3.2, we know that

$$\mathbb{E}\left[x'_{u,i} - x_{u,i}\right] = 0 \tag{35}$$

On the other hand, using the mean aggregator function, for any node $v \in \mathcal{V}$ and any dimension $i \in \{1, 2, \ldots, d\}$, we have:

$$(\mathbf{h}_{\mathcal{N}(v)})_i = \frac{1}{|\mathcal{N}(v)|} \sum_{u \in \mathcal{N}(v)} x_{u,i}$$
$$(\widehat{\mathbf{h}}_{\mathcal{N}(v)})_i = \frac{1}{|\mathcal{N}(v)|} \sum_{u \in \mathcal{N}(v)} x'_{u,i} \tag{36}$$

Considering 34 to 36 and using the Bernstein inequality, we have:

$$\begin{aligned}
&\Pr\left[\left|(\widehat{\mathbf{h}}_{\mathcal{N}(v)})_i - (\mathbf{h}_{\mathcal{N}(v)})_i\right| \geq \lambda\right] \\
&= \Pr\left[\left|\sum_{u \in \mathcal{N}(v)} (x'_{u,i} - x_{u,i})\right| \geq \lambda |N(v)|\right] \\
&\leq 2 \cdot \exp\left\{-\frac{\lambda^2 |\mathcal{N}(v)|}{\frac{2}{|\mathcal{N}(v)|}\sum_{u \in \mathcal{N}(v)} Var[x'_{u,i} - x_{u,i}] + \frac{2}{3}\lambda(\frac{\beta-\alpha}{2} + C)}\right\} \\
&= 2 \cdot \exp\left\{-\frac{\lambda^2 |\mathcal{N}(v)|}{2Var[x'_{u,i}] + \frac{2}{3}\lambda(\frac{\beta-\alpha}{2} + C)}\right\}
\end{aligned} \tag{37}$$

We can rewrite the variance of $x'_{u,i}$ in terms of C as:

$$Var[x'_{u,i}] = \frac{m}{d} C^2 - \left(x_{v,i} - \frac{\alpha+\beta}{2}\right)^2 \tag{38}$$

The asymptotic expressions involving ϵ are evaluated in $\epsilon \to 0$, which yields:

$$C = \frac{d(\beta-\alpha)}{2m} O(\frac{m}{\epsilon}) = O(\frac{d}{\epsilon}) \tag{39}$$

and therefore we have:

$$Var[x'_{u,i}] = \frac{m}{d}\left(O(\frac{d}{\epsilon})\right)^2 - \left(x_{v,i} - \frac{\alpha+\beta}{2}\right)^2 = O(\frac{md}{\epsilon^2}) \tag{40}$$

Substituting (39) and (40) in (37), we have:

$$\Pr\left[\left|(\widehat{\mathbf{h}}_{\mathcal{N}(v)})_i - (\mathbf{h}_{\mathcal{N}(v)})_i\right| \geq \lambda\right] \leq 2 \cdot \exp\left\{-\frac{\lambda^2 |\mathcal{N}(v)|}{O(\frac{md}{\epsilon^2}) + \lambda O(\frac{d}{\epsilon})}\right\}$$

According to the union bound, we have:

$$\begin{aligned}
&\Pr\left[\max_{i \in \{1,\ldots,d\}} \left|(\widehat{\mathbf{h}}_{\mathcal{N}(v)})_i - (\mathbf{h}_{\mathcal{N}(v)})_i\right| \geq \lambda\right] \\
&= \bigcup_{i=1}^{d} \Pr\left[\left|(\widehat{\mathbf{h}}_{\mathcal{N}(v)})_i - (\mathbf{h}_{\mathcal{N}(v)})_i\right| \geq \lambda\right] \\
&\leq \sum_{i=1}^{d} \Pr\left[\left|(\widehat{\mathbf{h}}_{\mathcal{N}(v)})_i - (\mathbf{h}_{\mathcal{N}(v)})_i\right| \geq \lambda\right] \\
&= 2d \cdot \exp\left\{-\frac{\lambda^2 |\mathcal{N}(v)|}{O(\frac{md}{\epsilon^2}) + \lambda O(\frac{d}{\epsilon})}\right\}
\end{aligned}$$

To ensure that $\max_{i \in \{1,\ldots,d\}} \left|(\widehat{\mathbf{h}}_{\mathcal{N}(v)})_i - (\mathbf{h}_{\mathcal{N}(v)})_i\right| < \lambda$ holds with at least $1 - \delta$ probability, it is sufficient to set

$$\delta = 2d \cdot \exp\left\{-\frac{\lambda^2 |\mathcal{N}(v)|}{O(\frac{md}{\epsilon^2}) + \lambda O(\frac{d}{\epsilon})}\right\} \tag{41}$$

Solving the above for λ, we get:

$$\lambda = O\left(\frac{\sqrt{d \log(d/\delta)}}{\epsilon\sqrt{|\mathcal{N}(v)|}}\right) \tag{42}$$

□

A.7 Corollary 3.7

Proof. The training steps in Algorithm 3.1 only process the output of the multi-bit encoder and the randomized response mechanism, which respectively provide ϵ_x-LDP and ϵ_y-LDP for each node. Private node features and labels are not used anywhere else in the algorithm except by the multi-bit encoder and the randomized response mechanism. Since Algorithm 3 calls the encoder and randomized response only once per node, and due to the basic composition theorem and the robustness of differentially private algorithms to post-processing [15], Algorithm 3 satisfies $(\epsilon_x + \epsilon_y)$-LDP for each node. □