

ProGAP: Progressive Graph Neural Networks with Differential Privacy Guarantees

Sina Sajadmanesh and Daniel Gatica-Perez

Idiap Research Institute, EPFL
{sajadmanesh,gatica}@idiap.ch

Abstract. Graph Neural Networks (GNNs) have become a popular tool for learning on graphs, but their widespread use raises privacy concerns as graph data can contain personal or sensitive information. Differentially private GNN models have been recently proposed to preserve privacy while still allowing for effective learning over graph-structured datasets. However, achieving an ideal balance between accuracy and privacy in GNNs remains challenging due to the intrinsic structural connectivity of graphs. In this paper, we propose a new differentially private GNN called ProGAP that uses a progressive training scheme to improve such accuracy-privacy trade-offs. Combined with the aggregation perturbation technique to ensure differential privacy, ProGAP splits a GNN into a sequence of overlapping submodels that are trained progressively, expanding from the first submodel to the complete model. Specifically, each submodel is trained over the privately aggregated node embeddings learned and cached by the previous submodels, leading to an increased expressive power compared to previous approaches while limiting the incurred privacy costs. We formally prove that ProGAP ensures edge-level and node-level privacy guarantees for both training and inference stages, and evaluate its performance on benchmark graph datasets. Experimental results demonstrate that ProGAP can achieve up to 5-10% higher accuracy than existing state-of-the-art differentially private GNNs.

Keywords: Graph Neural Network · Differential Privacy · Progressive Learning · Node Classification.

1 Introduction

Graph Neural Networks (GNNs) have emerged as a powerful tool for learning from graph-structured data, and their popularity has surged due to their ability to achieve impressive performance in a wide range of applications, including social network analysis, drug discovery, recommendation systems, and traffic prediction [2, 5, 14, 22, 48]. GNNs excel at learning from the structural connectivity of graphs by iteratively updating node embeddings through information aggregation and transformation from neighboring nodes, making them well-suited for tasks such as node classification, graph classification, and link prediction [7, 16, 25, 46, 50, 51]. However, as with many data-driven approaches,

GNNs can expose individuals to privacy risks when applied to graph data containing sensitive information, such as social connections, medical records, and financial transactions [36, 42]. Recent studies have shown that various attacks, such as link stealing, membership inference, and node attribute inference, can successfully break the privacy of graph datasets [18, 19, 34, 44], posing a significant challenge for the practical use of GNNs in privacy-sensitive applications.

To address the privacy concerns associated with GNNs, researchers have recently studied *differential privacy (DP)*, a well-established mathematical framework that provides strong privacy guarantees, usually by adding random noise to the data [9, 10]. However, applying DP to GNNs is very challenging due to the complex structural connectivity of graphs, rendering traditional private learning methods, such as differentially private stochastic gradient descent (DP-SGD) [1], infeasible [3, 8, 39]. Recently, the *aggregation perturbation (AP)* approach [39] has emerged as a state-of-the-art technique for ensuring DP in GNNs. Rather than perturbing the model gradients as done in the standard DP-SGD algorithm and its variants, this method perturbs the aggregate information obtained from the GNN neighborhood aggregation step. Consequently, such perturbations can obfuscate the presence of a single edge, which is called *edge-level privacy*, or a single node and all its adjacent edges, referred to as *node-level privacy* [37].

The key limitation of AP is its incompatibility with standard GNN architectures due to the high privacy costs it entails [39]. This is because conventional GNN models constantly query the aggregation functions with every update to the model parameters, which necessitates the re-perturbation of all aggregate outputs at every training iteration to ensure DP, leading to a significant increase in privacy costs. To mitigate this issue, Sajadmanesh *et al.* [39] proposed a method called GAP, which decouples the aggregation steps from the model parameters. In GAP, node features are recursively aggregated first, and then a classifier is learned over the resulting perturbed aggregations, enabling DP to be maintained without incurring excessive privacy costs. Due to having non-trainable aggregations, however, such decoupling approaches reduce the expressiveness of the GNN [12], leading to suboptimal accuracy-privacy trade-offs.

In the face of these challenges, we present a novel differentially private GNN, called “**Progressive GNN with Aggregation Perturbation**” (PROGAP). Our new method uses the same AP technique as in GAP to ensure DP. However, instead of decoupling the aggregation steps from the learnable modules, PROGAP adopts a multi-stage, progressive training paradigm to surmount the formidable privacy costs associated with AP. Specifically, PROGAP converts a K -layer GNN model into a sequence of overlapping submodels, where the i -th submodel comprises the first i layers of the model, followed by a lightweight supervision head layer with softmax activation that utilizes node labels to guide the submodel’s training. Starting with the shallowest submodel, PROGAP then proceeds progressively to train deeper submodels, each of which is referred to as a training stage. At every stage, the learned node embeddings from the preceding stage are aggregated, perturbed, and then cached to save privacy budget, allowing PROGAP to learn

a new set of private node embeddings. Ultimately, the last stage’s embeddings are used to generate final node-wise predictions.

The proposed progressive training approach overcomes the high privacy costs of AP by allowing the perturbations to be applied only once per stage rather than at every training iteration. PROGAP also maintains a higher level of expressive power compared to GAP, as the aggregation steps now operate on the learned embeddings from the preceding stages, which are more expressive than the raw node features. Moreover, we prove that PROGAP retains all the benefits of GAP, such as edge- and node-level privacy guarantees and zero-cost privacy at inference time. We evaluate PROGAP on five node classification datasets, including Facebook, Amazon, and Reddit, and demonstrate that it can achieve up to 10.4% and 5.5% higher accuracy compared to GAP under edge- and node-level DP with an epsilon of 1 and 8, respectively.

2 Related Work

Several recent studies have investigated differential privacy (DP) to provide formal privacy guarantees in various GNN learning settings. For example, Sajadmanesh and Gatica-Perez [38] propose a locally private GNN for a distributed learning environment, where node features and labels remain private, while the GNN training is federated by a central server with access to graph edges. Lin *et al.* [29] also introduce a locally private GNN, called SOLITUDE, that preserves edge privacy in a decentralized graph, where each node keeps its own private connections. However, both of these approaches use local differential privacy [24], which operates under a different problem setting from our method.

Other approaches propose edge-level DP algorithms for GNNs. Wu *et al.* [44] developed an edge-level private method that modifies the input graph directly through randomized response or the Laplace mechanism, followed by training a GNN on the resulting noisy graph. In contrast, Kolluri *et al.* [27] propose LPGNET, which adopts a tailored neural network architecture. Instead of directly using the graph edges, they encode graph adjacency information in the form of low-sensitivity cluster vectors, which are then perturbed using the Laplace mechanism to preserve edge-level privacy. Unlike our approach, however, neither of these methods provides node-level privacy guarantees.

Olatunji *et al.* [33] propose the first node-level private GNN by adapting the framework of PATE [35]. In their approach, a student GNN model is trained on public graph data, with each node privately labeled using teacher GNN models that are trained exclusively for the corresponding query node. Nevertheless, their approach relies on public graph data and may not be applicable in all situations. Daigavane *et al.* [8] extend the standard DP-SGD algorithm and privacy amplification by subsampling to bounded-degree graph data to achieve node-level DP, but their method fails to provide inference privacy. Finally, Sajadmanesh *et al.* [39] propose GAP, a private GNN learning framework that provides both edge-level and node-level privacy guarantees using the aggregation perturbation approach. They decouple the aggregation steps from the neu-

ral network model to manage the privacy costs of their method. Although our method leverages the same aggregation perturbation technique, we take a different approach to limit the privacy costs using a progressive training scheme.

The main concept behind progressive learning is to train the model on simpler tasks first and then gradually move towards more challenging tasks. It was originally introduced to stabilize the training of deep learning models and has been widely adopted in various computer vision applications, such as facial attribute editing [45], image super-resolution [43], image synthesis [23], and representation learning [28]. This technique has also been extended to federated learning, mainly to minimize the communication overhead between clients and the central server [4, 17, 41]. However, the potential benefit of progressive learning in DP applications has not been explored yet. In this paper, we are first to examine the advantages of progressive learning in the context of private GNNs.

3 Background

3.1 Differential Privacy

Differential privacy (DP) is a widely accepted framework for measuring the privacy guarantees of algorithms that operate on sensitive data. The main idea of DP is to ensure that the output of an algorithm is not significantly affected by the presence or absence of any particular individual’s data in the input. This means that even if an attacker has access to all but one individual’s data, they cannot determine whether that individual’s data was used in the computation. The formal definition of DP is as follows [10]:

Definition 1. *Given $\epsilon > 0$ and $\delta \in [0, 1]$, a randomized algorithm \mathcal{A} satisfies (ϵ, δ) -differential privacy, if for all adjacent datasets \mathcal{D} and \mathcal{D}' differing by at most one record and for all possible subsets of \mathcal{A} ’s outputs $S \subseteq \text{Range}(\mathcal{A})$:*

$$\Pr[\mathcal{A}(\mathcal{D}) \in S] \leq e^\epsilon \Pr[\mathcal{A}(\mathcal{D}') \in S] + \delta.$$

To adapt the definition of DP for graphs, two different notions of adjacency are defined: edge-level and node-level adjacency. In the former, two graphs are adjacent if they differ only in the presence of a single edge, whereas in the latter, the two graphs differ by a single node with its features, labels, and all attached edges. Accordingly, the definitions of edge-level and node-level DP are derived from these definitions [37]. Specifically, an algorithm \mathcal{A} provides edge-/node-level (ϵ, δ) -DP if for every two edge-/node-level adjacent graph datasets \mathcal{G} and \mathcal{G}' and any set of outputs $S \subseteq \text{Range}(\mathcal{A})$, we have $\Pr[\mathcal{A}(\mathcal{G}) \in S] \leq e^\epsilon \Pr[\mathcal{A}(\mathcal{G}') \in S] + \delta$.

3.2 Graph Neural Networks

Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with a set of nodes $\mathcal{V} = \{v_1, \dots, v_N\}$ and edges \mathcal{E} represented by an adjacency matrix $\mathbf{A} \in \{0, 1\}^{N \times N}$. Node features are represented by a matrix $\mathbf{X} \in \mathbb{R}^{N \times d}$, where \mathbf{X}_i denotes the d -dimensional feature

vector of node v_i . A common K -layer GNN is composed of K layers of graph convolution that are applied sequentially. Specifically, layer k takes as input the adjacency matrix \mathbf{A} and the node embeddings produced by layer $k - 1$, denoted by $\mathbf{X}^{(k-1)}$, and outputs a new embedding for each node by aggregating the embeddings of its adjacent neighbors, followed by a neural network transformation. In its simplest form, the formal update rule for layer k can be written as follows:

$$\mathbf{X}^{(k)} = \text{MLP} \left(\text{AGG}(\mathbf{A}, \mathbf{X}^{(k-1)}); \Theta^{(k)} \right), \quad (1)$$

where AGG is a differentiable permutation-invariant *neighborhood aggregation function* and MLP denotes a multilayer perceptron parameterized by $\Theta^{(k)}$ that takes the aggregated embeddings as input and produces a new embedding for each node. The aggregation function can take various forms, such as mean, sum, or max pooling. The input to the first layer is $\mathbf{X}^{(0)} = \mathbf{X}$, i.e., the initial node features. The output of the final layer $\mathbf{X}^{(K)}$ can then be used for downstream tasks, such as node classification or link prediction.

3.3 Problem Definition

Consistent with prior work [8, 39], we focus on the node classification task. Consider a GNN-based node classification model $\mathcal{M}(\mathbf{A}, \mathbf{X}; \Theta)$ parameterized by a set of parameters Θ that takes the adjacency matrix \mathbf{A} and the node features \mathbf{X} , and outputs the corresponding predicted node labels $\hat{\mathbf{Y}}$:

$$\hat{\mathbf{Y}} = \mathcal{M}(\mathbf{A}, \mathbf{X}; \Theta). \quad (2)$$

We seek to minimize a standard classification loss function \mathcal{L} , such as cross-entropy, with respect to the set of model parameters Θ :

$$\arg \min_{\Theta} \mathcal{L}(\mathcal{M}(\mathbf{A}, \mathbf{X}; \Theta), \mathbf{Y}), \quad (3)$$

where $\mathbf{Y} \in \{0, 1\}^{N \times C}$ is the ground-truth node labels with C being the number of classes. Given a graph dataset $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X}, \mathbf{Y})$, our goal is to ensure the privacy of \mathcal{G} at both the training (Eq. 3) and inference (Eq. 2) phases of the model \mathcal{M} , using the differential privacy notions defined for graphs, i.e., edge-level and node-level DP. Note that preserving privacy during the inference stage is of utmost importance since the adjacency information of the graph is still used at inference time to generate the predicted labels, and thus sensitive information about the graph could potentially be leaked even with Θ being differentially private [39].

4 Proposed Method

In this section, we present our proposed PROGAP method, which leverages the aggregation perturbation (AP) technique [39] to ensure differential privacy but introduces a novel progressive learning scheme to restrain the privacy costs of AP incurred during training. The overview of PROGAP architecture is illustrated in Figure 1, and its forward propagation (inference) and training algorithms are presented in Algorithm 1 and Algorithm 2, respectively. In the following, we first describe our method in detail and then analyze its privacy guarantees.

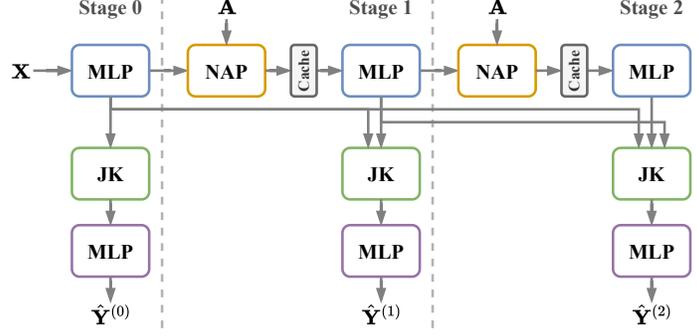


Fig. 1. An example PROGAP architecture with three stages. MLP and JK represent multi-layer perceptron and Jumping Knowledge [47] modules, respectively. NAP denotes the normalize-aggregate-perturb module used to ensure the privacy of the adjacency matrix, with its output cached immediately after computation to save privacy budget. Training is done progressively, starting with the first stage and then expanding to the second and third stages, each using its own head MLP. The final prediction is obtained by the head MLP of the last stage.

4.1 Model Architecture and Training

We start by considering a simple non-private sequential GNN model \mathcal{M} with K aggregation layers as the following:

$$\mathbf{X}^{(0)} = \text{MLP}_{base}^{(0)}(\mathbf{X}; \Theta_{base}^{(0)}), \quad (4)$$

$$\mathbf{X}^{(k)} = \text{MLP}_{base}^{(k)}(\text{AGG}(\mathbf{A}, \mathbf{X}^{(k-1)}); \Theta_{base}^{(k)}), \quad \forall k \in \{1, \dots, K\}, \quad (5)$$

$$\hat{\mathbf{Y}} = \text{MLP}_{head}(\mathbf{X}^{(K)}; \Theta_{head}), \quad (6)$$

where $\mathbf{X}^{(k)}$ is the node embeddings generated at layer k by $\text{MLP}_{base}^{(k)}$ having parameters $\Theta_{base}^{(k)}$, and MLP_{head} is a multi-layer perceptron parameterized by Θ_{head} with the softmax activation function that maps the final embeddings $\mathbf{X}^{(K)}$ to the predicted class probabilities $\hat{\mathbf{Y}}$.

To make this model differentially private, we follow the aggregation perturbation technique proposed by Sajadmanesh *et al.* [39] and add noise to the output of the aggregation function. Specifically, we replace the original aggregation function AGG in Eq. 5 with a *Normalize-Aggregate-Perturb* mechanism defined as:

$$\text{NAP}(\mathbf{A}, \mathbf{X}; \sigma) = \left[\sum_{j=1}^N \frac{\mathbf{X}_i}{\|\mathbf{X}_i\|_2} \mathbf{A}_{j,i} + \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_d) \mid \forall i \in \{1, \dots, N\} \right], \quad (7)$$

where N is the number of nodes, d is the dimension of the input node embeddings, and σ is the standard deviation of the Gaussian noise. Concretely, the

Algorithm 1: PROGAP Forward Propagation $\mathcal{M}_s(\mathbf{A}, \mathbf{X}; \sigma, \mathfrak{P}_s)$

Input : Stage s , adjacency matrix \mathbf{A} ; node features \mathbf{X} ; noise standard deviation σ ; model parameters $\mathfrak{P}_s = \bigcup_{k=0}^s \{\boldsymbol{\Theta}_{base}^{(k)}\} \cup \{\boldsymbol{\Theta}_{jump}^{(s)}, \boldsymbol{\Theta}_{head}^{(s)}\}$

Output : Predicated node labels $\hat{\mathbf{Y}}^{(s)}$

- 1 $\tilde{\mathbf{X}}^{(0)} \leftarrow \mathbf{X}$
- 2 **for** $k \in \{0, \dots, s\}$ **do**
- 3 **if** $k > 0$ *and* $\tilde{\mathbf{X}}^{(k)}$ *is not cached* **then**
- 4 $\tilde{\mathbf{X}}^{(k)} \leftarrow \text{NAP}(\mathbf{A}, \tilde{\mathbf{X}}^{(k-1)}; \sigma)$
- 5 Cache $\tilde{\mathbf{X}}^{(k)}$
- 6 **end**
- 7 $\mathbf{X}^{(k)} \leftarrow \text{MLP}_{base}^{(k)}(\tilde{\mathbf{X}}^{(k)}; \boldsymbol{\Theta}_{base}^{(k)})$
- 8 **end**
- 9 $\hat{\mathbf{Y}}^{(s)} \leftarrow \text{MLP}_{head}^{(s)}(\text{JK}^{(s)}(\{\mathbf{X}^{(0)}, \dots, \mathbf{X}^{(s)}\}; \boldsymbol{\Theta}_{jump}^{(s)}); \boldsymbol{\Theta}_{head}^{(s)})$
- 10 **return** $\hat{\mathbf{Y}}^{(s)}$

NAP mechanism row-normalizes the input embeddings to limit the contribution of each node to the aggregated output, then applies the sum aggregation function followed by adding Gaussian noise to the results.

It can be easily shown that the resulting model provides edge-level DP as every query to the adjacency matrix \mathbf{A} is immediately perturbed with noise. However, training such a model comes at the cost of a significant increase in the privacy budget, which is proportional to the number of queries to the adjacency matrix. Concretely, with T training iterations, the NAP mechanism is queried KT times (at each forward pass and each layer), leading to an excessive accumulated privacy cost of $O(\sqrt{KT})$.

To reduce this cost, we propose a progressive training approach as the following: We first split the model \mathcal{M} into $K + 1$ overlapping submodels, where submodel \mathcal{M}_s , $s \in \{0, 1, \dots, K\}$, is defined as:

$$\tilde{\mathbf{X}}^{(s)} = \text{NAP}(\mathbf{A}, \mathbf{X}^{(s-1)}; \sigma), \quad (8)$$

$$\mathbf{X}^{(s)} = \text{MLP}_{base}^{(s)}(\tilde{\mathbf{X}}^{(s)}; \boldsymbol{\Theta}_{base}^{(s)}), \quad (9)$$

$$\hat{\mathbf{Y}}^{(s)} = \text{MLP}_{head}^{(s)}\left(\text{JK}^{(s)}\left(\bigcup_{k=0}^s \{\mathbf{X}^{(k)}\}; \boldsymbol{\Theta}_{jump}^{(s)}\right); \boldsymbol{\Theta}_{head}^{(s)}\right), \quad (10)$$

where $\tilde{\mathbf{X}}^{(s)}$ is the noisy aggregate embeddings of \mathcal{M}_s , with $\tilde{\mathbf{X}}^{(0)} = \mathbf{X}$. $\text{JK}^{(s)}$ is a Jumping Knowledge module [47] with parameters $\boldsymbol{\Theta}_{jump}^{(s)}$ that combines the embeddings generated by submodels \mathcal{M}_0 to \mathcal{M}_s , and $\text{MLP}_{head}^{(s)}$ is a lightweight, 1-layer head MLP with parameters $\boldsymbol{\Theta}_{head}^{(s)}$ used to train \mathcal{M}_s . Finally, $\hat{\mathbf{Y}}^{(s)}$ is the output predictions of \mathcal{M}_s . Then, we progressively train the model in $K + 1$ stages, starting from the shallowest submodel \mathcal{M}_0 and gradually expanding to

Algorithm 2: PROGAP Training

Input : Adjacency matrix \mathbf{A} ; node features \mathbf{X} ; node labels \mathbf{Y} ; model depth K ; noise standard deviation σ ;

Output : Trained model parameters \mathfrak{P}_K^*

- 1 initialize $\Theta_{base}^{(0)}, \Theta_{jump}^{(0)}, \Theta_{head}^{(0)}$ randomly
- 2 $\mathfrak{P}_0 \leftarrow \{\Theta_{base}^{(0)}, \Theta_{jump}^{(0)}, \Theta_{head}^{(0)}\}$
- 3 **for** $s \in \{0, \dots, K\}$ **do**
- 4 $\mathfrak{P}_s^* \leftarrow \arg \min_{\mathfrak{P}} \mathcal{L}(\mathcal{M}_s(\mathbf{A}, \mathbf{X}; \sigma, \mathfrak{P}), \mathbf{Y})$
- 5 **if** $s < K$ **then**
- 6 initialize $\Theta_{base}^{(s+1)}, \Theta_{jump}^{(s+1)}, \Theta_{head}^{(s+1)}$ randomly
- 7 $\mathfrak{P}_{s+1}^* \leftarrow \mathfrak{P}_s^* \cup \{\Theta_{base}^{(s+1)}, \Theta_{jump}^{(s+1)}, \Theta_{head}^{(s+1)}\} \setminus \{\Theta_{jump}^{*(s)}, \Theta_{head}^{*(s)}\}$
- 8 **end**
- 9 **end**
- 10 **return** \mathfrak{P}_K^*

the deepest submodel \mathcal{M}_K (which is equivalent to the full model \mathcal{M}) as explained by Algorithm 2. For the final inference after training, we simply use the labels predicted by the last submodel \mathcal{M}_K , i.e., $\hat{\mathbf{Y}} = \hat{\mathbf{Y}}^{(K)}$.

The key point in this training strategy is that we immediately save the outputs of NAP modules on their first query and reuse them throughout the training. More specifically, at each stage s , the perturbed aggregate embedding matrix $\tilde{\mathbf{X}}^{(s)}$ computed in the first forward pass of \mathcal{M}_s (via Eq. 8) is stored in the cache and reused in all further queries. This caching mechanism allows us to reduce the privacy costs of the model by a factor of T , as the NAP module in this case is only queried K times (once per stage) instead of KT times. At the same time, the aggregations $\tilde{\mathbf{X}}^{(s)}$ are computed over the embeddings $\mathbf{X}^{(s-1)}$ that are already learned in the preceding stage $s-1$, which provide more expressive power than the raw node features as they also encode information from the adjacency matrix and node labels, and thus lead to better performance.

Remark 1. The proposed PROGAP model can also be trained in a layerwise fashion, i.e., by training each layer $\text{MLP}_{base}^{(k)}$ individually, while keeping the parameters of the preceding layers frozen and using the same caching mechanism. Note that this is different from the proposed progressive approach, in which all the parameters from layer 0 to layer s , i.e., $\Theta_{base}^{(0)}, \dots, \Theta_{base}^{(s)}$ are trained together in each stage s . In Section 6, we show that such a progressive training strategy leads to better performance than layerwise training.

4.2 Privacy Analysis

With the following theorem, we show that the proposed training strategy provides edge-level DP. The proof is provided in Appendix A.2.

Theorem 1. *Given the maximum stage $K \geq 0$ and noise variance σ^2 , for any $\delta \in (0, 1)$ Algorithm 2 satisfies edge-level (ϵ, δ) -DP with $\epsilon = \frac{K}{2\sigma^2} + \sqrt{2K \log(1/\delta)}/\sigma$.*

To ensure node-level DP, however, we must train every submodel using DP-SGD or its variants, as in this case node features and labels are also private and can be leaked with non-private training. Theorem 2 establishes the node-level DP guarantee of PROGAP’s training algorithm when combined with DP-SGD:

Theorem 2. *Given the number of nodes N , batch-size $B < N$, number of per-stage training iterations T , gradient clipping threshold $C > 0$, maximum stage $K \geq 0$, maximum cut-off degree $D \geq 1$, noise variance for aggregation perturbation $\sigma_{AP}^2 > 0$, and noise variance for gradient perturbation $\sigma_{GP}^2 > 0$, Algorithm 2 satisfies node-level (ϵ, δ) -DP for any $\delta \in (0, 1)$ with:*

$$\begin{aligned} \epsilon \leq \min_{\alpha > 1} & \frac{(K+1)T}{\alpha-1} \log \left\{ \left(1 - \frac{B}{N}\right)^{\alpha-1} \left(\alpha \frac{B}{N} - \frac{B}{N} + 1\right) \right. \\ & + \binom{\alpha}{2} \left(\frac{B}{N}\right)^2 \left(1 - \frac{B}{N}\right)^{\alpha-2} e^{-\frac{C^2}{2\sigma_{GP}^2}} \\ & \left. + \sum_{l=3}^{\alpha} \binom{\alpha}{l} \left(1 - \frac{B}{N}\right)^{\alpha-l} \left(\frac{B}{N}\right)^l e^{-(l-1)\left(\frac{C^2 l}{2\sigma_{GP}^2}\right)} \right\} \\ & + \frac{DK\alpha}{2\sigma_{AP}^2} + \frac{\log(1/\delta)}{\alpha-1}, \end{aligned}$$

providing that the optimization in line 4 of Algorithm 2 is done using DP-SGD.

The proof is deferred to Appendix A.3. Note that to decrease the node-level sensitivity of the NAP mechanism (i.e., the impact of adding/removing a node on the output of the NAP mechanism), we assume an upper bound D on node degrees, and randomly sample edges from the graph to ensure that each node has no more than D outgoing edges. This is a standard technique to ensure bounded-degree graphs [8, 39].

In addition to training privacy, PROGAP also guarantees privacy during inference at both edge and node levels without any further privacy costs. This is because the entire noisy aggregate matrices $\tilde{\mathbf{X}}^{(i)}$ corresponding to all the nodes –both training and test ones– are already computed and cached during training and reused for inference (i.e., lines 4 and 5 of Algorithm 1 is not executed at inference time). As a result, the inference for a node no longer depends on its private neighborhood and is done by post-processing differentially private outputs, which does not incur any additional privacy costs.

5 Experimental Setup

We test our proposed method on node-wise classification tasks and evaluate its effectiveness in terms of classification accuracy and privacy guarantees.

Table 1. Dataset Statistics.

DATASET	# NODES	# EDGES	# FEATURES	# CLASSES	MED. DEGREE
FACEBOOK	26,406	2,117,924	501	6	62
REDDIT	116,713	46,233,380	602	8	209
AMAZON	1,790,731	80,966,832	100	10	22
FACEBOOK-100	1,120,280	86,304,478	537	6	57
WENET	37,576	22,684,206	44	4	286

5.1 Datasets

We conduct experiments on three real-world datasets that have been used in previous work [8, 33, 39], namely Facebook [40], Reddit [16], and Amazon [6], and also two new datasets: Facebook-100 [40] and WeNet [13, 30]. The Facebook dataset is a collection of anonymized social network data from UIUC students, where nodes represent users, edges indicate friendships, and the task is to predict students’ class year. The Reddit dataset comprises a set of Reddit posts as nodes, where edges represent if the same user commented on both posts, and the goal is to predict the posts’ subreddit. The Amazon dataset is a product co-purchasing network, with nodes representing products and edges indicating if two products are purchased together, and the objective is to predict product category. Facebook-100 is an extended version of the Facebook dataset combining the social network of 100 different American universities. WeNet is a mobile sensing dataset collected from university students in four different countries. Nodes represent eating events, which are linked based on the similarity of location and Wi-Fi sensor readings. Node features are extracted based on cellular and application sensors, and the goal is to predict the country of the events. A summary of the datasets is provided in Table 1.

5.2 Baselines

We compare our PROGAP method against GAP [39], which is the closest related work to ours. We use GAP’s official implementation on GitHub¹ and follow the same experimental setup as reported in the original paper. We do not include other available differentially private GNN approaches as they either: (i) are outperformed by GAP (e.g., [8, 44]) or (ii) have different problem settings (e.g., [34, 38]) that make them not directly comparable to our method.

5.3 Implementation Details

We use PyTorch Geometric [11] for implementing the models, `autodp`² for privacy accounting, and Opacus [49] for DP training. We follow the same experimental setup as GAP [39], and randomly split the nodes in all the datasets into

¹ <https://github.com/sisaman/GAP>

² <https://github.com/yuxiangw/autodp>

Table 2. Comparison of Experimental Results (Mean Accuracy \pm 95% CI)

PRIVACY LEVEL	METHOD	ϵ	FACEBOOK	REDDIT	AMAZON	FACEBOOK-100	WENET
NON-PRIVATE	PROGAP	∞	84.5 \pm 0.24	99.3 \pm 0.03	93.3 \pm 0.04	74.4 \pm 0.14	73.9 \pm 0.25
	GAP	∞	80.5 \pm 0.42	99.5 \pm 0.01	92.0 \pm 0.10	66.4 \pm 0.35	69.7 \pm 0.14
EDGE LEVEL	PROGAP	1.0	77.2 \pm 0.33	97.8 \pm 0.05	84.2 \pm 0.07	56.9 \pm 0.30	68.8 \pm 0.23
	GAP	1.0	69.4 \pm 0.39	97.5 \pm 0.06	78.8 \pm 0.26	46.5 \pm 0.58	62.4 \pm 0.28
NODE LEVEL	PROGAP	8.0	69.3 \pm 0.33	94.0 \pm 0.04	79.1 \pm 0.10	48.5 \pm 0.36	61.0 \pm 0.34
	GAP	8.0	63.9 \pm 0.49	93.9 \pm 0.09	77.6 \pm 0.07	43.0 \pm 0.20	58.2 \pm 0.39

training, validation, and test sets with 75/10/15% ratio, respectively. We vary ϵ within $\{0.25, 0.5, 1, 2, 4, \infty\}$ for the edge-level privacy ($\epsilon = \infty$ corresponds to the non-private setting) and within $\{2, 4, 8, 16, 32\}$ for the node-level privacy setting. For each ϵ value, we tune the following hyperparameters based on the mean validation set accuracy computed over 10 runs: MLP_{base} layers in $\{1, 2\}$, model depth K in $\{1, 2, 3, 4, 5\}$, and learning rate in $\{0.01, 0.05\}$. The value of δ is fixed per each dataset to be smaller than the inverse number of private units (i.e., edges for edge-level privacy, nodes for node-level privacy). For all cases, we set the number of MLP_{head} layers to 1 and use concatenation for the JK modules. Additionally, we set the number of hidden units to 16 and use the SeLU activation function [26]. We use batch normalization except for the node-level setting, for which we use group normalization with one group. Under the edge-level setting, we train the models with full-sized batches for 100 epochs using the Adam optimizer and perform early stopping based on the validation set accuracy. For the node-level setting, we use randomized neighbor sampling to bound the maximum degree D to 50 for Amazon, 100 for Facebook and Facebook-100, and 400 for Reddit and WeNet. We use DP-Adam [15] with a clipping threshold of 1.0. We tune the number of per-stage epochs in $\{5, 10\}$ and set the batch size to 256, 1024, 2048, 4096, and 4096 for Facebook, Reddit, Amazon, Facebook-100, and WeNet, respectively. Finally, we report the average test accuracy over 10 runs with 95% confidence intervals calculated by bootstrapping with 1000 samples. We open-source our implementation on GitHub.³

6 Results and Discussion

6.1 Accuracy-Privacy Trade-off

Table 2 presents the test accuracy of PROGAP against GAP at three different privacy levels: non-private with $\epsilon = \infty$, edge-level privacy with $\epsilon = 1$, and node-level privacy with $\epsilon = 8$. The results are reported as mean accuracy \pm 95% confidence interval. We observe that PROGAP outperforms GAP in almost all cases, and often by a substantial margin. Specifically, in the non-private setting,

³ It will be made public upon acceptance.

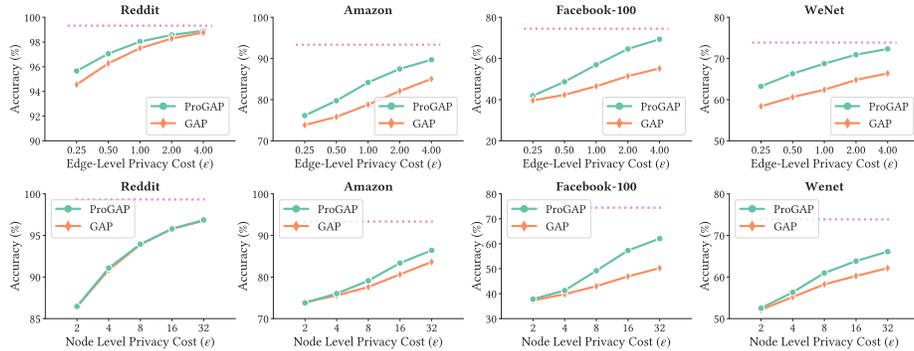


Fig. 2. Accuracy-privacy trade-off of edge-level (top) and node-level (bottom) private methods. The dotted line represents the accuracy of the non-private ProGAP.

PROGAP achieves significantly higher test accuracies on all datasets except Reddit, on which GAP performs only slightly better. Under both the edge-level and node-level privacy settings, however, PROGAP consistently outperforms GAP on all datasets, with the largest performance gap of 10.4% and 5.5% accuracy points, respectively, which are both observed on Facebook-100.

To examine the performance of the methods at different privacy budgets, we varied ϵ between 0.25 to 4 for edge-level privacy and 2 to 32 for node-level private algorithms. We then recorded the accuracy of each method for each privacy budget. The outcome for both edge-level and node-level privacy settings is depicted in Figure 2.⁴ Notably, we observe that PROGAP achieves higher accuracies than GAP across all ϵ values tested and approaches the non-private accuracy more quickly under both privacy settings. This is because in PROGAP each aggregation step is computed on the node embeddings learned in the previous stage, providing greater expressive power than GAP, which recursively computes the aggregations on the initial node representations.

6.2 Convergence Analysis

We examine the convergence of PROGAP to further understand its behavior under the two privacy settings. We report the training and validation accuracy of PROGAP per training step under edge-level privacy with $\epsilon = 1$ and node-level privacy with $\epsilon = 8$. For all datasets, PROGAP is trained for 100 and 10 epochs per stage under edge and node-level privacy, respectively. We fix $K = 5$ in all settings. The results are shown in Figure 3. We observe that both training and validation accuracies increase as PROGAP moves from stage 0 to 5, with diminishing returns for more stages, which indicates the higher importance of the nearby neighbors to each node, since the receptive field of nodes grows with the number of stages. Moreover, we observe negligible discrepancies between training and validation accuracy when the model converges, which suggests higher

⁴ The results on the Facebook dataset are omitted due to space limitation.

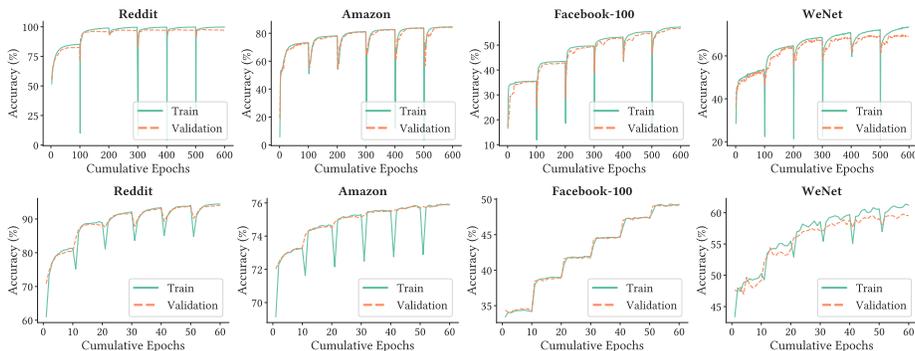


Fig. 3. Convergence of PROGAP with $K = 5$ under edge-level (top) and node-level (bottom) privacy, with $\epsilon = 1$ and $\epsilon = 8$, respectively.

resilience to privacy attacks, such as membership inference, which typically rely on large generalization gaps. This result is in line with previous work showing the effectiveness of DP against privacy attacks [20, 21, 32, 39].

6.3 Effect of the Model Depth

We explore how the performance of PROGAP is influenced by modifying the model depth K , or equivalently, the number of stages $K + 1$. We experiment with different values of K ranging from 1 to 5 and evaluate PROGAP’s accuracy under varying privacy budgets of $\epsilon \in \{0.25, 1, 4\}$ for edge-level DP and $\epsilon \in \{2, 8, 32\}$ for node-level privacy. The results are demonstrated in Figure 4. We observe that PROGAP can generally gain advantages from increasing the depth, but there is a compromise depending on the privacy budget: deeper models lead to better accuracy under higher privacy budgets, while lower privacy budgets require shallower models to achieve optimal performance. This is because PROGAP can leverage data from more remote nodes with a higher value of K , which can boost the final accuracy, but it also increases the amount of noise in the aggregations, which has a detrimental effect on the model’s accuracy. When the privacy budget is lower and the amount of noise is greater, PROGAP has the best performance at smaller values of K . But as the privacy budget grows, the magnitude of the noise is lowered, enabling the models to take advantage of greater K values.

6.4 Progressive vs. Layerwise Training

We compare the performance of PROGAP using two different training strategies: progressive training and layerwise training. Similar to Table 2, we report the test accuracy of both strategies at three different privacy levels: non-private with $\epsilon = \infty$, edge-level privacy with $\epsilon = 1$, and node-level privacy with $\epsilon = 8$. The results are presented in Table 3. Overall, we observe that the progressive training yields higher accuracies than the layerwise strategy in most cases, which as mentioned in Section 4, is due to the higher capacity of the progressive approach.

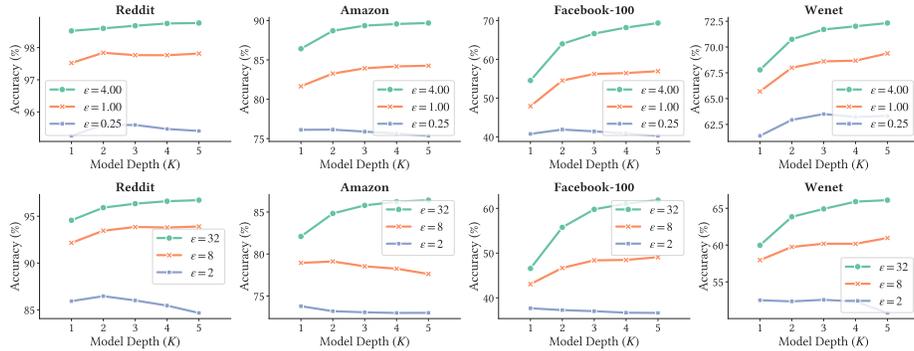


Fig. 4. Effect of the model depth on PROGAP’s accuracy under edge-level (top) and node-level (bottom) privacy.

Table 3. Accuracy Comparison of Progressive (PR) and Layerwise (LW) Training

PRIVACY LEVEL	ϵ	TRAINING STRATEGY	FACEBOOK	REDDIT	AMAZON	FACEBOOK-100	WENET
NON-PRIVATE	∞	PR	84.5 ± 0.24	99.3 ± 0.03	93.3 ± 0.04	74.4 ± 0.14	73.9 ± 0.25
		LW	85.6 ± 0.29	99.3 ± 0.03	92.9 ± 0.04	74.0 ± 0.16	71.9 ± 0.19
EDGE LEVEL	1.0	PR	77.2 ± 0.33	97.8 ± 0.05	84.2 ± 0.07	56.9 ± 0.30	68.8 ± 0.23
		LW	76.8 ± 0.22	98.0 ± 0.06	83.4 ± 0.08	55.7 ± 0.25	67.7 ± 0.25
NODE LEVEL	8.0	PR	69.3 ± 0.33	94.0 ± 0.04	79.1 ± 0.10	48.5 ± 0.36	61.0 ± 0.34
		LW	68.7 ± 0.48	94.0 ± 0.07	78.8 ± 0.05	49.2 ± 0.57	59.3 ± 0.42

7 Conclusion

In this paper, we introduced PROGAP, a novel differentially private GNN that improves the challenging accuracy-privacy trade-off in learning from graph data. Our approach uses a progressive training scheme that splits the GNN into a sequence of overlapping submodels, each of which is trained over privately aggregated node embeddings learned and cached by the previous submodels. By combining this technique with the aggregation perturbation method, we formally proved that PROGAP can ensure edge-level and node-level privacy guarantees for both training and inference stages. Empirical evaluations on benchmark graph datasets demonstrated that PROGAP can achieve state-of-the-art accuracy by outperforming existing methods. Future work could include exploring new architectures or training strategies to further improve the accuracy-privacy trade-off of differentially private GNNs, especially in the more challenging node-level privacy setting.

Acknowledgments

This work was supported by the European Commission’s H2020 Program ICT-48-2020, AI4Media Project, under grant number 951911. It was also supported by the European Commission’s H2020 WeNet Project, under grant number 823783.

References

1. Abadi, M., Chu, A., Goodfellow, I., McMahan, H.B., Mironov, I., Talwar, K., Zhang, L.: Deep learning with differential privacy. In: Proceedings of the 2016 ACM SIGSAC conference on computer and communications security. pp. 308–318 (2016)
2. Ahmedt-Aristizabal, D., Armin, M.A., Denman, S., Fookes, C., Petersson, L.: Graph-based deep learning for medical diagnosis and analysis: Past, present and future. arXiv preprint arXiv:2105.13137 (2021)
3. Ayle, M., Schuchardt, J., Gosch, L., Zügner, D., Günnemann, S.: Training differentially private graph neural networks with random walk sampling. arXiv preprint arXiv:2301.00738 (2023)
4. Belilovsky, E., Eickenberg, M., Oyallon, E.: Decoupled greedy learning of CNNs. In: III, H.D., Singh, A. (eds.) Proceedings of the 37th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 119, pp. 736–745. PMLR (13–18 Jul 2020)
5. Cheung, M., Moura, J.M.F.: Graph neural networks for covid-19 drug discovery. In: 2020 IEEE International Conference on Big Data (Big Data). pp. 5646–5648 (2020). <https://doi.org/10.1109/BigData50022.2020.9378164>
6. Chiang, W.L., Liu, X., Si, S., Li, Y., Bengio, S., Hsieh, C.J.: Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 257–266 (2019)
7. Corso, G., Cavalleri, L., Beaini, D., Liò, P., Veličković, P.: Principal neighbourhood aggregation for graph nets. In: Advances in Neural Information Processing Systems (2020)
8. Daigavane, A., Madan, G., Sinha, A., Thakurta, A.G., Aggarwal, G., Jain, P.: Node-level differentially private graph neural networks. arXiv preprint arXiv:2111.15521 (2021)
9. Dwork, C.: Differential privacy: A survey of results. In: International conference on theory and applications of models of computation. pp. 1–19. Springer (2008)
10. Dwork, C., McSherry, F., Nissim, K., Smith, A.: Calibrating noise to sensitivity in private data analysis. In: Theory of cryptography conference. pp. 265–284. Springer (2006)
11. Fey, M., Lenssen, J.E.: Fast graph representation learning with PyTorch Geometric. In: ICLR Workshop on Representation Learning on Graphs and Manifolds (2019)
12. Fey, M., Lenssen, J.E., Weichert, F., Leskovec, J.: Gnnautoscale: Scalable and expressive graph neural networks via historical embeddings. In: International Conference on Machine Learning. pp. 3294–3304. PMLR (2021)
13. Giunchiglia, F., Bison, I., Busso, M., Chenu-Abente, R., Rodas, M., Zeni, M., Gunel, C., Veltri, G., De Götzen, A., Kun, P., et al.: A worldwide diversity pilot on daily routines and social practices (2020) (2021)

14. Gkalelis, N., Goulas, A., Galanopoulos, D., Mezaris, V.: Objectgraphs: Using objects and a graph convolutional network for the bottom-up recognition and explanation of events in video. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 3375–3383 (2021)
15. Gylberth, R., Adnan, R., Yazid, S., Basaruddin, T.: Differentially private optimization algorithms for deep neural networks. In: 2017 International Conference on Advanced Computer Science and Information Systems (ICACSIS). pp. 387–394. IEEE (2017)
16. Hamilton, W.L., Ying, R., Leskovec, J.: Inductive representation learning on large graphs. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. pp. 1025–1035 (2017)
17. He, C., Li, S., Soltanolkotabi, M., Avestimehr, S.: Pipetransformer: Automated elastic pipelining for distributed training of transformers. arXiv preprint arXiv:2102.03161 (2021)
18. He, X., Jia, J., Backes, M., Gong, N.Z., Zhang, Y.: Stealing links from graph neural networks. In: 30th {USENIX} Security Symposium ({USENIX} Security 21) (2021)
19. He, X., Wen, R., Wu, Y., Backes, M., Shen, Y., Zhang, Y.: Node-level membership inference attacks against graph neural networks. arXiv preprint arXiv:2102.05429 (2021)
20. Jagielski, M., Ullman, J.R., Oprea, A.: Auditing differentially private machine learning: How private is private sgd? In: Proceedings of the Advances in Neural Information Processing (NeurIPS). Virtual Event (December 2020)
21. Jayaraman, B., Evans, D.: Evaluating differentially private machine learning in practice. In: 28th USENIX Security Symposium (USENIX Security 19). pp. 1895–1912. USENIX Association, Santa Clara, CA (Aug 2019), <https://www.usenix.org/conference/usenixsecurity19/presentation/jayaraman>
22. Jiang, W., Luo, J.: Graph neural network for traffic forecasting: A survey. Expert Systems with Applications p. 117921 (2022)
23. Karras, T., Aila, T., Laine, S., Lehtinen, J.: Progressive growing of gans for improved quality, stability, and variation. arXiv preprint arXiv:1710.10196 (2017)
24. Kasiviswanathan, S.P., Lee, H.K., Nissim, K., Raskhodnikova, S., Smith, A.: What can we learn privately? *SIAM Journal on Computing* **40**(3), 793–826 (2011)
25. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations (ICLR) (2017)
26. Klambauer, G., Unterthiner, T., Mayr, A., Hochreiter, S.: Self-normalizing neural networks. In: Proceedings of the 31st international conference on neural information processing systems. pp. 972–981 (2017)
27. Kolluri, A., Baluta, T., Hooi, B., Saxena, P.: Lpgnet: Link private graph networks for node classification. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. p. 1813–1827. CCS '22, Association for Computing Machinery, New York, NY, USA (2022). <https://doi.org/10.1145/3548606.3560705>
28. Li, Z., Murkute, J.V., Gyawali, P.K., Wang, L.: Progressive learning and disentanglement of hierarchical representations. arXiv preprint arXiv:2002.10549 (2020)
29. Lin, W., Li, B., Wang, C.: Towards private learning on decentralized graphs with local differential privacy. *IEEE Transactions on Information Forensics and Security* **17**, 2936–2946 (2022). <https://doi.org/10.1109/TIFS.2022.3198283>

30. Meegahapola, L., Droz, W., Kun, P., de Götzen, A., Nutakki, C., Diwakar, S., Correa, S.R., Song, D., Xu, H., Bidoglia, M., et al.: Generalization and personalization of mobile sensing-based mood inference models: An analysis of college students in eight countries. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* **6**(4), 1–32 (2023)
31. Mironov, I.: Rényi differential privacy. In: 2017 IEEE 30th Computer Security Foundations Symposium (CSF). pp. 263–275. IEEE (2017)
32. Nasr, M., Song, S., Thakurta, A., Papernot, N., Carlini, N.: Adversary instantiation: Lower bounds for differentially private machine learning. In: *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*. San Francisco, CA, USA (May 2021)
33. Olatunji, I.E., Funke, T., Khosla, M.: Releasing graph neural networks with differential privacy guarantees. *arXiv preprint arXiv:2109.08907* (2021)
34. Olatunji, I.E., Nejd, W., Khosla, M.: Membership inference attack on graph neural networks. In: 2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA). pp. 11–20. IEEE (2021)
35. Papernot, N., Abadi, M., Erlingsson, U., Goodfellow, I., Talwar, K.: Semi-supervised knowledge transfer for deep learning from private training data. *arXiv preprint arXiv:1610.05755* (2016)
36. Qiu, J., Tang, J., Ma, H., Dong, Y., Wang, K., Tang, J.: Deepinf: Social influence prediction with deep learning. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. pp. 2110–2119 (2018)
37. Raskhodnikova, S., Smith, A.: Differentially private analysis of graphs. *Encyclopedia of Algorithms* (2016)
38. Sajadmanesh, S., Gatica-Perez, D.: Locally private graph neural networks. In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. pp. 2130–2145 (2021)
39. Sajadmanesh, S., Shamsabadi, A.S., Bellet, A., Gatica-Perez, D.: Gap: Differentially private graph neural networks with aggregation perturbation. In: *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, Anaheim, CA (Aug 2023)
40. Traud, A.L., Mucha, P.J., Porter, M.A.: Social structure of facebook networks. *Physica A: Statistical Mechanics and its Applications* **391**(16), 4165–4180 (2012)
41. Wang, H.P., Stich, S., He, Y., Fritz, M.: ProgFed: Effective, communication, and computation efficient federated learning by progressive training. In: *International Conference on Machine Learning*. pp. 23034–23054. PMLR (2022)
42. Wang, J., Zhang, S., Xiao, Y., Song, R.: A review on graph neural network methods in financial applications. *arXiv preprint arXiv:2111.15367* (2021)
43. Wang, Y., Perazzi, F., McWilliams, B., Sorkine-Hornung, A., Sorkine-Hornung, O., Schroers, C.: A fully progressive approach to single-image super-resolution. In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. pp. 864–873 (2018)
44. Wu, F., Long, Y., Zhang, C., Li, B.: Linkteller: Recovering private edges from graph neural networks via influence analysis. *arXiv preprint arXiv:2108.06504* (2021)
45. Wu, R., Zhang, G., Lu, S., Chen, T.: Cascade ef-gan: Progressive facial expression editing with local focuses. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. pp. 5021–5030 (2020)

46. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? In: International Conference on Learning Representations (2019), <https://openreview.net/forum?id=ryGs6iA5Km>
47. Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.i., Jegelka, S.: Representation learning on graphs with jumping knowledge networks. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 5453–5462. PMLR, Stockholmsmässan, Stockholm Sweden (10–15 Jul 2018)
48. Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W.L., Leskovec, J.: Graph convolutional neural networks for web-scale recommender systems. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 974–983 (2018)
49. Yousefpour, A., Shilov, I., Sablayrolles, A., Testuggine, D., Prasad, K., Malek, M., Nguyen, J., Ghosh, S., Bharadwaj, A., Zhao, J., Cormode, G., Mironov, I.: Opacus: User-friendly differential privacy library in PyTorch. arXiv preprint arXiv:2109.12298 (2021)
50. Zhang, M., Chen, Y.: Link prediction based on graph neural networks. Advances in Neural Information Processing Systems **31**, 5165–5175 (2018)
51. Zhang, M., Li, P., Xia, Y., Wang, K., Jin, L.: Labeling trick: A theory of using graph neural networks for multi-node representation learning. Advances in Neural Information Processing Systems **34** (2021)
52. Zhu, Y., Wang, Y.X.: Poission subsampled rényi differential privacy. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proceedings of the 36th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 97, pp. 7634–7642. PMLR (09–15 Jun 2019), <https://proceedings.mlr.press/v97/zhu19c.html>

ProGAP: Progressive Graph Neural Networks with Differential Privacy Guarantees

Supplementary Material

Sina Sajadmanesh and Daniel Gatica-Perez

Idiap Research Institute, EPFL
{sajadmanesh,gatica}@idiap.ch

A Deferred Theoretical Arguments

A.1 Background: Rényi Differential Privacy

The proofs presented in this section are based on *Rényi Differential Privacy* (RDP) [31], which is an alternative definition of DP that gives tighter sequential composition results. The formal definition of RDP is as follows:

Definition 1 (Rényi Differential Privacy [31]). *Given $\alpha > 1$ and $\epsilon > 0$, a randomized algorithm \mathcal{A} satisfies (α, ϵ) -RDP if for every adjacent datasets X and X' , we have:*

$$D_\alpha(\mathcal{A}(X)\|\mathcal{A}(X')) \leq \epsilon, \quad (1)$$

where $D_\alpha(P\|Q)$ is the Rényi divergence of order α between probability distributions P and Q defined as:

$$D_\alpha(P\|Q) = \frac{1}{\alpha - 1} \log \mathbb{E}_{x \sim Q} \left[\frac{P(x)}{Q(x)} \right]^\alpha.$$

A key property of RDP is that it can be converted to standard (ϵ, δ) -DP using the Proposition 1 of [31], as follows:

Proposition 1 (From RDP to (ϵ, δ) -DP [31]). *If \mathcal{A} is an (α, ϵ) -RDP algorithm, then it also satisfies $(\epsilon + \log(1/\delta)/\alpha - 1, \delta)$ -DP for any $\delta \in (0, 1)$.*

A.2 Proof of Theorem 1

Proof. In Algorithm 2, the graph’s adjacency is only used when the NAP mechanism is invoked during the forward propagation of submodels \mathcal{M}_1 to \mathcal{M}_K . According to Lemma 1 of [39], the edge-level sensitivity of the NAP mechanism is 1, and thus based on Corollary 3 of [31], each individual query to the NAP mechanism is $(\alpha, \alpha/2\sigma^2)$ -RDP. Due to ProGAP’s caching system, the NAP mechanism is only invoked K times during training (once for each submodel), and the rest of the training process does not query the graph edges. As a result, Algorithm 2 can be seen as an adaptive composition of K NAP mechanisms, which based on Proposition 1 of [31], is $(\alpha, K\alpha/2\sigma^2)$ -RDP. According to Proposition 3 of [31], this is equivalent to edge-level (ϵ, δ) -DP with $\epsilon = \frac{K\alpha}{2\sigma^2} + \frac{\log(1/\delta)}{\alpha - 1}$. Minimizing this expression over $\alpha > 1$ gives $\epsilon = \frac{K}{2\sigma^2} + \sqrt{2K \log(1/\delta)}/\sigma$.

A.3 Proof of Theorem 2

Proof. Algorithm 2 is composed of $K + 1$ stages, where each stage $s \in \{1, \dots, K\}$ starts by computing and perturbing the aggregate embeddings (Eq. 8), which is the only part where the graph adjacency information is involved. As this part is privatized by the NAP mechanism, the rest of the process in stage $s \geq 1$ is just normal graph-agnostic training over tabular-like data, which is made private using DP-SGD. The exception is stage 0, which does not use the graph's adjacency information at all, and thus it is just privatized using DP-SGD. Therefore, Algorithm 2 can be seen as an adaptive composition of K NAP mechanisms and $K + 1$ DP-SGD algorithms. According to Lemma 3 of [39], the NAP mechanism is node-level $(\alpha, D\alpha/2\sigma_{AP}^2)$ -RDP. The DP-SGD algorithm itself is a composition of T subsampled Gaussian mechanisms, which according to Theorem 11 of [52] and Proposition 1 of [31] is $(\alpha, \epsilon_{\text{DPSGD}})$ -RDP, where:

$$\begin{aligned} \epsilon_{\text{DPSGD}} \leq & \frac{T}{\alpha - 1} \log \left\{ \left(1 - \frac{B}{N}\right)^{\alpha-1} \left(\alpha \frac{B}{N} - \frac{B}{N} + 1\right) \right. \\ & + \binom{\alpha}{2} \left(\frac{B}{N}\right)^2 \left(1 - \frac{B}{N}\right)^{\alpha-2} e^{\frac{C^2}{\sigma_{GP}^2}} \\ & \left. + \sum_{l=3}^{\alpha} \binom{\alpha}{l} \left(1 - \frac{B}{N}\right)^{\alpha-l} \left(\frac{B}{N}\right)^l e^{(l-1)\left(\frac{C^2 l}{2\sigma_{GP}^2}\right)} \right\}. \end{aligned}$$

Overall, according to Proposition 1 of [31], the composition of K NAP mechanisms and $K + 1$ DP-SGD algorithms is $(\alpha, \epsilon_{\text{total}})$ -RDP, where:

$$\begin{aligned} \epsilon_{\text{total}} \leq & \frac{(K+1)T}{\alpha - 1} \log \left\{ \left(1 - \frac{B}{N}\right)^{\alpha-1} \left(\alpha \frac{B}{N} - \frac{B}{N} + 1\right) \right. \\ & + \binom{\alpha}{2} \left(\frac{B}{N}\right)^2 \left(1 - \frac{B}{N}\right)^{\alpha-2} e^{\frac{C^2}{\sigma_{GP}^2}} \\ & \left. + \sum_{l=3}^{\alpha} \binom{\alpha}{l} \left(1 - \frac{B}{N}\right)^{\alpha-l} \left(\frac{B}{N}\right)^l e^{(l-1)\left(\frac{C^2 l}{2\sigma_{GP}^2}\right)} \right\} \\ & + \frac{DK\alpha}{2\sigma_{AP}^2}. \end{aligned}$$

The proof is completed by applying Proposition 3 of [31] to the above expression and minimizing the upper bound over $\alpha > 1$:

$$\begin{aligned} \epsilon \leq \min_{\alpha > 1} & \frac{(K+1)T}{\alpha-1} \log \left\{ \left(1 - \frac{B}{N}\right)^{\alpha-1} \left(\alpha \frac{B}{N} - \frac{B}{N} + 1\right) \right. \\ & + \binom{\alpha}{2} \left(\frac{B}{N}\right)^2 \left(1 - \frac{B}{N}\right)^{\alpha-2} e^{\frac{C^2}{2\sigma_{GP}^2}} \\ & \left. + \sum_{l=3}^{\alpha} \binom{\alpha}{l} \left(1 - \frac{B}{N}\right)^{\alpha-l} \left(\frac{B}{N}\right)^l e^{(l-1)\left(\frac{C^2 l}{2\sigma_{GP}^2}\right)} \right\} \\ & + \frac{DK\alpha}{2\sigma_{AP}^2} + \frac{\log(1/\delta)}{\alpha-1}, \end{aligned}$$