EPFL

Thèse n° 10 245

Privacy-Preserving Machine Learning on Graphs

Présentée le 1er décembre 2023

Faculté des sciences et techniques de l'ingénieur
Laboratoire de l'IDIAP
Programme doctoral en génie électrique

pour l'obtention du grade de Docteur ès Sciences

par

Sina SAJADMANESH

Acceptée sur proposition du jury

Dr D. Gillet, président du jury
Prof. D. Gatica-Perez, directeur de thèse
Prof. H. Haddadi, rapporteur
Prof. E. De Cristofaro, rapporteur
Prof. K. Aberer, rapporteur

École
polytechnique
fédérale
de Lausanne

2023

The roots of education are bitter,
but the fruit is sweet.
— Aristotle

To my dear parents and my loving wife, whose unwavering affection, support, and encouragement throughout this academic journey illuminated my path...

Acknowledgements

I would like to express my heartfelt gratitude to the following individuals and institutions who have been instrumental in the completion of my PhD thesis. First, I am deeply indebted to my advisor, Prof. Daniel Gatica-Perez, for his unwavering support, patience, and invaluable insights throughout this journey. Second, I extend my sincere appreciation to the members of my examining committee: Dr. Denis Gillet, Prof. Karl Aberer, Prof. Emiliano De Cristofaro, and Prof. Hamed Haddadi. Their constructive feedback and support were instrumental in shaping this research. Third, I acknowledge the financial support provided by the Swiss National Science Foundation (Dusk2Dawn Project) and the European Commission (AI4Media Project). Their generous support allowed me to conduct this research and push the boundaries of knowledge in my field. Fourth, I am deeply grateful to my fellow researchers and collaborators, both within and outside my institution, for their camaraderie, brainstorming sessions, and contributions that enriched the research experience. Fifth, I would like to thank all my friends and colleagues at Idiap Research Institute and EPFL who stood by me, offering motivation, support, laughter, and sometimes a much-needed break from academic pursuits. Last but not least, I express my deepest gratitude to my family for their unwavering love and support throughout my life. Their sacrifices and understanding made this journey possible.

This thesis represents the culmination of years of hard work and dedication from many individuals and organizations. I am humbled by the collective support and encouragement that has propelled me forward in my academic pursuits.

Thank you all for your unwavering belief in me and for being an integral part of this academic journey.

Lausanne, September 19, 2023 Sina Sajadmanesh

Abstract

Graph Neural Networks (GNNs) have emerged as a powerful tool for learning on graphs, demonstrating exceptional performance in various domains. However, as GNNs become increasingly popular, new challenges arise. One of the most pressing is the need to ensure privacy when learning from graphs since they often contain personal information like social interactions and financial transactions. Balancing the preservation of privacy with the effectiveness of GNNs is a crucial yet challenging task due to the unique relational nature of graphs, which sets them apart from other data types like images and text. This thesis presents an in-depth exploration of privacy-preserving GNNs, explicitly focusing on Differential Privacy (DP) as a rigorous mathematical framework for formalizing the privacy guarantees of machine learning algorithms.

First, we focus on local DP and propose a privacy-preserving Graph Convolutional Network (GCN) model, allowing a central server to efficiently communicate with the graph nodes to privately collect their features and estimate the first-layer graph convolution of the GNN. Building on the same foundations, we then propose an architecture-agnostic approach for settings where all the node-specific data, including features and labels, are private and locally stored on the nodes, but the edges remain public to the central server that trains the GNN. Our contributions include a locally private randomizer for perturbing and compressing node features, a denoising mechanism based on graph convolution to alleviate the noise effect, and a robust learning algorithm to deal with noisy labels.

Next, we shift our focus to global DP, where the entire graph data is considered private, and the goal is to train and subsequently release a GNN model with DP guarantees. Our main contribution, which enables private learning and inference on graph data, is the aggregation perturbation (AP) technique, which adds noise to GNNs' neighborhood aggregation step to achieve DP. Built upon this technique, we first propose a tailored GNN architecture that decouples the aggregation steps from the model parameters to reduce the privacy cost of AP, leading to a significant decrease in the privacy costs of the model. Then, we propose a progressive GNN model to further improve the accuracy-privacy trade-off of the AP technique without sacrificing privacy. The new method iteratively builds GNNs using private node embeddings from prior steps, maintaining their representational power and managing privacy costs.

Abstract

Empirical evaluations demonstrate that the proposed approaches provide strong privacy guarantees while effectively maintaining the utility of the learned models. Our methods can be applied in a wide range of applications, from social network analysis to disease prediction, thereby having a broad impact on both academia and industry. By addressing the challenge of privacy in learning on graphs, this thesis contributes to the broader efforts in promoting responsible and ethical use of machine learning and provides a solid foundation for future research on trustworthy GNNs, paving the way for their adoption in various domains where privacy is paramount.

Keywords: *Graph Neural Networks, Differential Privacy, Privacy-Preserving Machine Learning, Graph Data, Progressive Learning, Node Classification, Link Prediction*

Résumé

Les réseaux de neurones en graphes (GNNs) ont émergé comme un outil puissant pour l'apprentissage sur les graphes, démontrant des performances exceptionnelles dans divers domaines. Cependant, avec la popularité croissante des GNNs, de nouveaux défis apparaissent. L'un des plus pressants est la nécessité d'assurer la confidentialité lors de l'apprentissage à partir de graphes, car ils contiennent souvent des informations personnelles. Cette thèse présente une exploration approfondie des GNNs préservant la vie privée, en se concentrant sur la Confidentialité Différentielle (DP) comme un cadre mathématique rigoureux pour formaliser les garanties de confidentialité des algorithmes d'apprentissage automatique.

Premièrement, nous nous concentrons sur la DP locale et proposons un modèle de réseau de convolution sur graphes (GCN) préservant la vie privée, permettant à un serveur central de communiquer avec les nœuds du graphe pour collecter leurs caractéristiques de manière privée et estimer la première convolution de graphe de la couche du GNN. En nous appuyant sur les mêmes fondements, nous proposons ensuite une approche indépendante de l'architecture pour des contextes où toutes les données spécifiques aux nœuds, y compris les caractéristiques et les labels, sont privées et stockées localement sur les nœuds, mais les arêtes restent publiques pour le serveur central qui entraîne le GNN. Nos contributions comprennent un randomiseur localement privé pour perturber et compresser les caractéristiques des nœuds, un mécanisme de dé bruitage basé sur la convolution de graphes pour atténuer l'effet du bruit et un algorithme d'apprentissage pour gérer les labels bruités.

Ensuite, nous orientons notre attention vers la DP globale, où l'ensemble des données du graphe est considéré comme privé et l'objectif est de former puis de publier un modèle GNN avec des garanties de DP. Notre principale contribution, qui permet un apprentissage et une inférence privés sur les données de graphes, est la technique de perturbation de l'agrégation (AP), qui ajoute du bruit à l'étape d'agrégation des voisins des GNNs pour atteindre la DP. Basé sur cette technique, nous proposons d'abord une architecture GNN sur mesure qui découple les étapes d'agrégation des paramètres du modèle pour réduire le coût de confidentialité de l'AP, ce qui entraîne une diminution significative du coût de confidentialité du modèle. Puis, nous proposons un modèle GNN progressif pour améliorer encore le compromis précision-confidentialité de la technique AP sans sacrifier la confidentialité. La nouvelle méthode construite de manière itérative des GNNs en utilisant les embeddings de nœuds privés des étapes précédentes, en conservant leur pouvoir de représentation et en gérant les

coûts de confidentialité.

Les évaluations empiriques démontrent que les approches proposées fournissent de robustes garanties de confidentialité tout en préservant efficacement l'utilité des modèles appris. Nos méthodes peuvent être appliquées dans un large éventail d'applications, ayant ainsi un large impact sur l'académie et l'industrie. En abordant le défi de la confidentialité dans l'apprentissage sur les graphes, cette thèse contribue aux efforts plus larges visant à promouvoir l'utilisation responsable et éthique de l'apprentissage automatique et fournit une base solide pour la recherche future sur les GNNs fiables, ouvrant la voie à leur adoption dans divers domaines où la confidentialité est primordiale.

Mots clefs : *Réseaux de Neurones en Graphes, Confidentialité Différentielle, Apprentissage Automatique Préservant la Vie Privée, Données de Graphes, Apprentissage Progressif, Classification de Nœuds, Prédiction de Liens*

Contents

List of Figures

List of Tables

1 Introduction

Graphs have become a ubiquitous form of data representation in numerous domains due to their ability to naturally encode relationships between entities (Newman, 2018). From social networks and citation networks to biological networks and the World Wide Web, graph data is everywhere. In a social network, for instance, individuals are represented as nodes and their interactions as edges, forming a complex web of relationships. Similarly, in a citation network, academic papers are nodes and citations are edges, providing a rich structure that reflects the development of scientific ideas. The prevalence of such graph-structured data across a wide range of applications underscores the need for effective tools to analyze and learn from them.

Given the ubiquity and importance of graph data, it is no surprise that machine learning has been extensively applied to graphs (Chami et al., 2022). However, traditional machine learning methods often struggle with graph data due to its complex and irregular structure (Battaglia et al., 2018). Unlike tabular data or image data, which have a fixed structure, graph data is inherently relational and lacks a grid-like topology. This makes it challenging to apply conventional machine learning techniques, which often rely on the assumption of independent and identically distributed data (Bishop & Nasrabadi, 2006). Despite these challenges, the potential rewards of successfully applying machine learning to graph data are immense: from predicting missing links in a social network (Liben-Nowell & Kleinberg, 2003) to identifying communities of similar nodes (Fortunato, 2010), machine learning can unlock valuable insights hidden in the structure and attributes of graphs.

To address the challenges of learning from graph data, a new class of machine learning models known as Graph Neural Networks (GNNs) has emerged (Scarselli et al., 2008), which extend traditional neural networks by incorporating mechanisms to handle the complex, non-Euclidean structure of graph data. These models operate by propagating and transforming feature information between connected nodes, enabling the capture of both local and global graph structures. They have proven to be the state-of-the-art tool for a variety of graph-based tasks, including node classification (Hamilton et al., 2017a; Kipf & Welling, 2017), link prediction (M. Zhang & Chen, 2018), and graph classification (Ying et al., 2018), and have found applications in diverse fields, including recommendation systems, bioinformatics,

and traffic prediction (Z. Wu et al., 2020). However, as GNNs become increasingly popular, new challenges arise. One of the most pressing is the need to ensure privacy when learning from graph data. This dissertation focuses on this critical issue, exploring novel methods for privacy-preserving machine learning on graphs.

1.1 Motivation

The increasing prevalence of graph data and the growing popularity of GNNs have led to a surge in privacy concerns. Graph data, due to its inherent structure and relational nature, often encapsulates sensitive information. For instance, in a social network graph, nodes could represent users, and edges could represent their friendships or interactions. Such data can contain personal details, preferences, behaviors, and other sensitive attributes of the individuals involved. Similarly, in bioinformatics, a graph could represent protein-protein interactions or genetic data, the disclosure of which could lead to serious privacy violations (Shringarpure & Bustamante, 2015). Therefore, the sensitivity of graph data underscores the importance of privacy preservation when dealing with such data.

Traditional machine learning models, when trained on sensitive data, have been shown to leak private information (Carlini et al., 2019; Fredrikson et al., 2015; Shokri et al., 2017). This leakage can occur at different stages of the machine learning pipeline, e.g., during the training process, where the model inadvertently learns and retains some of the sensitive information present in the training data. For example, a machine learning model trained on medical records could potentially reveal a patient's health status (Shokri et al., 2017), while a model trained on text data could inadvertently expose personal identifiers or sensitive conversations (Carlini et al., 2019). Even when data is anonymized, sophisticated techniques can sometimes re-identify individuals based on their data patterns (Narayanan & Shmatikov, 2008). This risk of privacy leakage in traditional machine learning models raises serious concerns when applied to graph data, given its sensitive nature.

Graph-structured data introduces additional privacy challenges due to its relational nature. Unlike traditional data, where each data point can be considered independently, graph data consists of nodes and edges that are interconnected. This means that the information about one node can potentially reveal information about its neighbors, leading to a higher risk of privacy breaches. Consequently, GNNs can inadvertently exacerbate these privacy issues, as they learn by propagating information between connected nodes. This process, while effective for learning the structure of the graph, can also lead to the leakage of sensitive information (Duddu et al., 2020). Researchers have shown that GNNs can be vulnerable to different kinds of privacy attacks, where an attacker can infer some sensitive information about a node, edge, or the entire graph based on the GNN's outputs (X. He, Jia, et al., 2021a; X. He, Wen, et al., 2021; Olatunji, Nejdl, & Khosla, 2021; F. Wu et al., 2021; Z. Zhang, Liu, Huang, Wang, et al., 2022; Z. Zhang et al., 2021a). For example, via the membership inference attack (Shokri et al., 2017), an attacker in a social network graph might be able to infer a user's private attributes,

such as their political affiliation or health status, based on their connections and the GNN's predictions (X. He, Wen, et al., 2021; Olatunji, Nejdl, & Khosla, 2021). Besides, the link stealing attack on a GNN trained on a social network designed for recommending friendships can unveil the existing connections between users (X. He, Jia, et al., 2021a; F. Wu et al., 2021). Similarly, a GNN trained on the social graph of individuals affected by COVID-19 can assist government authorities in forecasting the disease's transmission patterns, but an adversary could potentially extract private information about the patients involved through the graph reconstruction attack (Z. Zhang, Liu, Huang, Wang, et al., 2022; Z. Zhang et al., 2021a).

The challenge, therefore, lies not just in developing machine learning methods that can effectively learn from graph data, but also in ensuring that these methods respect the privacy of individuals, which is crucial for fostering trust and promoting the responsible use of machine learning on graph data. Motivated by this challenge, this thesis proposes novel methods for privacy-preserving machine learning on graphs, aiming to strike a balance between the utility of GNNs and the privacy of individuals. This balance is crucial in ensuring that we can leverage the power of GNNs for various applications, without compromising privacy.

1.2 Challenges

The task of preserving privacy in machine learning on graphs, particularly in the context of Graph Neural Networks (GNNs), is a complex and multifaceted problem. While there has been a surge of interest in privacy-preserving machine learning, most of the existing work has focused on traditional machine learning models, such as deep neural networks (DNNs). However, GNNs are fundamentally different from these models, and therefore, the privacy risks associated with them are also different, leading to new challenges. The unique characteristics of graph data, including its relational nature and the diversity of its elements, introduce a range of challenges that are not present in traditional privacy-preserving machine learning. These challenges are further compounded by the fact that many common privacy-preserving techniques are not directly applicable to graph data. This section outlines the primary challenges that must be addressed in order to develop effective privacy-preserving machine learning techniques for graphs.

Privacy in Different Graph Elements. Graphs are rich data types composed of various elements, including node features, node labels, edges, and more, each of which can carry sensitive information. For instance, node features might represent personal details, preferences, or behaviors. In a social network graph, these features could include a user's interests, activities, or demographic information. Node labels, on the other hand, could indicate classifications or groups, e.g., political affiliation or health status. Edges might signify relationships or interactions, such as friendships in a social network or transactions in a financial network. Ensuring that the process of learning from these various graph elements does not expose this sensitive information is a significant challenge.

Utility-Privacy Trade-off. There is often a trade-off between the utility of a machine learning model and the level of privacy it provides. Striking a balance between maintaining the utility of GNNs and ensuring privacy is a significant challenge. Ensuring privacy often involves introducing noise or obfuscation, which can degrade the performance of the model. For example, adding noise to the node features or the graph structure can help preserve privacy, but it can also make it more difficult for the GNN to learn meaningful patterns from the data. Similarly, obfuscating certain graph elements can protect privacy, but it can also limit the GNN's ability to leverage these elements for learning. Finding ways to preserve privacy without significantly compromising the utility of GNNs is a key challenge.

Applicability of Common Privacy-Preserving Techniques. Common privacy-preserving machine learning practices, such as differential privacy (Dwork, 2008), federated learning (T. Li et al., 2019), split learning (Vepakomma et al., 2018), homomorphic encryption (Sathya et al., 2018), and secure multiparty computation (Evans et al., 2018), which were originally designed for standard data types, such as images and text, cannot be trivially applied to graphs. The unique structure and relational nature of graph data present additional complexities that these techniques are not inherently equipped to handle. Unlike structured data types, where each data point can be considered independently, graph data consists of interconnected nodes and edges. This interconnectedness means that even if an individual node's data is protected, information can still be exposed through its connections. For instance, secure multiparty computation, which allows multiple parties to compute a function over their inputs while keeping these inputs private, assumes that the computation can be split across different parties. However, this assumption is challenging to satisfy for GNNs, which involve computations (e.g., message passing) that depend on the entire graph. Similarly, differential privacy, which provides a mathematical framework for quantifying privacy risks, faces challenges when applied to graph data. The addition of noise to maintain DP can disrupt the structure of the graph, affecting the performance of GNNs (Kolluri et al., 2022). Moreover, the propagation of information between connected nodes in GNNs can lead to the leakage of sensitive information, voiding the privacy guarantees of standard differentially private model training algorithms (Daigavane et al., 2021). These examples illustrate the need for developing new, graph-specific privacy-preserving techniques.

These challenges underscore the complexity of the problem this thesis aims to address. Our contributions, which are introduced in the following section, are designed to tackle these challenges, offering promising directions for future research in privacy-preserving machine learning on graphs.

1.3 Contributions and Outline

This thesis presents a series of substantial contributions to the field of privacy-preserving machine learning on graphs, specifically focusing on the integration of differential privacy

(DP) with graph neural networks (GNNs).

Differential privacy is a robust privacy framework that provides a mathematical guarantee of privacy, making it a powerful tool for privacy-preserving machine learning. It allows for the quantification of privacy loss and provides a mechanism to control the trade-off between privacy and utility. Differential privacy operates under two models: local and global. In the local model of DP, privacy is ensured at the level of individual data points. Each data point is perturbed independently before being transmitted to the server. For example, if the data is about user locations, rather than sending the precise GPS coordinates, the location would be blurred to a nearby point. This methodology provides strong privacy guarantees, but it often comes with the disadvantage of increased noise and thus, decreased data utility (Kasiviswanathan et al., 2011). On the other hand, the global model of DP ensures privacy at the level of the entire dataset. Here, the raw data is utilized to compute a result, and only then is noise introduced to this result before its release. An example could be a company wanting to release aggregate statistics about its user base without revealing sensitive information about any specific individual. The raw data (without noise) is used to calculate the statistic, and then noise is added to that calculated result. This method provides a more suitable balance between privacy protection and data utility, compared to the local model (Dwork, Roth, et al., 2014).

The first two contributions of this thesis are based on the local model of DP, while the latter two contributions leverage the global model of DP. These contributions represent substantial advancements in the development of privacy-preserving techniques for machine learning on graphs, providing promising directions for future research in this field.

The thesis is organized into seven main chapters, including this introductory chapter. The subsequent chapters each present a unique approach to integrating DP with GNNs, discuss the broader implications of this work, and conclude the thesis with future directions:

Chapter 2: When Differential Privacy Meets Graph Neural Networks. This chapter introduces a novel privacy-preserving GNN learning framework that combines the power of Graph Convolutional Networks (GCNs) with Local Differential Privacy (LDP). The framework is designed to address the problem of node attribute privacy in GNNs, where individual nodes in the graph have potentially sensitive attributes that need to be protected. The proposed framework allows a central server to efficiently communicate with the graph nodes to privately collect their features and estimate the first-layer graph convolution of the GNN. The chapter provides a comprehensive theoretical analysis of the proposed method, including its formal privacy guarantee and error bound. It also presents extensive experimental results over several real-world graph datasets, demonstrating the effectiveness of the proposed method for both privacy-preserving node classification and link prediction tasks.

Chapter 3: Locally Private Graph Neural Networks. Building on the foundation laid in Chapter 2, this chapter presents LPGNN, a novel approach to privacy-preserving GNNs that leverages LDP. LPGNN introduces a new noise injection mechanism and a novel graph convolution operation that together ensure the privacy of individual nodes during the learning process. It also introduces specific mechanisms to learn GNN models with private node labels. The chapter provides a detailed theoretical analysis of the privacy-utility trade-off in LPGNN and validates the effectiveness of the proposed method through extensive experiments on real-world datasets. The results demonstrate that LPGNN can effectively protect the privacy of individual nodes while still allowing the GNN to learn meaningful patterns from the data.

Chapter 4: Private Graph Neural Networks with Aggregation Perturbation. This chapter introduces GAP, a novel approach to privacy-preserving GNNs that leverages the global model of DP. GAP introduces a new aggregation perturbation technique that injects noise directly into the aggregation process of GNNs. This approach allows GAP to provide stronger privacy guarantees compared to LDP-based methods while maintaining competitive performance. The chapter provides a comprehensive theoretical analysis of GAP's privacy guarantees and demonstrates its effectiveness through extensive experiments on several benchmark datasets. The results show that GAP can effectively balance the trade-off between privacy and utility in GNNs.

Chapter 5: Progressive Learning on Graphs with Differential Privacy. This chapter builds upon the work of Chapter 4 by introducing ProGAP, a progressive learning framework for GNNs that also leverages the global model of DP. ProGAP introduces a novel progressive learning strategy that allows for the gradual refinement of the GNN's parameters while maintaining privacy guarantees. The chapter provides a detailed theoretical analysis of ProGAP's privacy guarantees and demonstrates its effectiveness through extensive experiments on several benchmark datasets. The results show that ProGAP can provide even stronger privacy guarantees than GAP while achieving higher utility.

Chapter 6: Conclusion and Future Directions. This final chapter summarizes the key findings of the thesis, discusses the implications of this work for the field of privacy-preserving machine learning on graphs, and suggests potential avenues for future research.

The significance of this dissertation lies in its contributions to the field of privacy-preserving machine learning on graphs, a burgeoning area of research that has far-reaching implications across numerous domains. The proposed methods provide strong privacy guarantees while maintaining the utility of the learned models, and can be applied in a wide range of applications, from social network analysis to disease prediction, thereby having a broad impact on both academia and industry. Furthermore, by focusing on differential privacy, this thesis contributes to the theoretical foundations of differentially private machine learning, paving the way for future research in this field. Lastly, by addressing the challenge of privacy in machine

learning on graphs, this thesis contributes to the broader efforts in promoting responsible and ethical use of machine learning. As machine learning models become increasingly integrated into our daily lives, ensuring privacy is not just a technical challenge but also a societal one. This dissertation, therefore, has implications beyond the technical realm, contributing to ongoing discussions about privacy and ethics in machine learning.

2 When Differential Privacy Meets Graph Neural Networks

As discussed in Chapter 1, Graph Neural Networks have proven to be exceptionally effective in deriving graph representations for a variety of downstream inferential tasks. However, when working with graph data, privacy issues can surface, particularly when the nodes represent people or human-related attributes encompassing personal information. While prior research has showcased diverse methods for privacy-preserving deep learning over non-relational data types, such as images, audio, video, and text, less attention has been paid to privacy challenges when deploying deep learning algorithms on graphs.

As a result, this chapter proposes the development of a privacy-preserving learning algorithm that offers formal privacy assurances for Graph Convolutional Networks (GCNs). Our method leverages Local Differential Privacy (LDP) to address the issue of node attribute privacy, where graph nodes possess potentially sensitive attributes that must remain private but could prove valuable for developing rich node representations within a centralized learning environment.

More specifically, we introduce an LDP algorithm that enables a central server to interface with graph nodes, privately gathering their data to estimate the graph convolution layer of a GCN. We delve into the theoretical attributes of our method and compare it with the state-of-the-art mechanisms. Experimental results, using real-world graph datasets, substantiate the efficacy of our proposed method for both privacy-preserving node classification and link prediction tasks, thereby validating our theoretical findings.

The structure of this chapter is as follows: In Section 2.1, we introduce the problem and motivation behind our work, and outline the main contributions of this chapter. Next, we review related works in Section 2.2, covering graph representation learning methods and privacy-preserving machine learning techniques. It is followed by Section 2.3, where we provide the necessary background on graph neural networks and local differential privacy. Then in Section 2.4, we delve into our privacy-preserving graph representation learning method addressing node attribute privacy. Section 2.5 presents the theoretical analysis of our method. In Section 2.6 and Section 2.7, we elaborate experimental setup and discuss results. Finally, in Section 2.8, we conclude the chapter and discuss potential future directions.

The work presented in this chapter is available as a preprint on ArXiv:

- Sajadmanesh, S., & Gatica-Perez, D. (2020). When differential privacy meets graph neural networks. *arXiv preprint arXiv:2006.05535v3*

2.1 Introduction

Following the advances of deep neural networks in various machine learning domains, such as computer vision, natural language understanding, and speech processing, in the past few years, extending deep learning models for graph-structured data has attracted growing interest, leading to the advent of Graph Neural Networks (GNNs) (Scarselli et al., 2008). These models have shown superior performance on a wide range of applications in social sciences (Hamilton et al., 2017a), biology (Rhee et al., 2017), molecular chemistry (Duvenaud et al., 2015), and so on, achieving state-of-the-art results in various graph-based tasks, such as node classification (Y. Zhang et al., 2019), link prediction (M. Zhang & Chen, 2018), and community detection (Z. Chen et al., 2017). However, the majority of real-world graphs linked to human-oriented actions, such as social and economic networks, commonly hold sensitive data that reveals personal aspects of individuals. In the context of a social network, details like a person's connections, profile content, preferences, and interactions can potentially hold varying degrees of privacy. Given the modern data protection policies, such as the General Data Protection Regulations (GDPR), the need to develop GNN models that respect privacy is critical, particularly for applications that operate on graph data consisting of users' private information.

2.1.1 Problem and Motivation

Considering the privacy concerns mentioned, we introduce the concept of node attribute privacy, where individual nodes in the graph are carrying potentially sensitive attributes that are kept confidential, while the remainder of the graph is visible from a central server's perspective. This server intends to use these private attributes to train a GNN across the graph. The ability to develop a graph representation using these private node features could be significantly beneficial in several specific uses. For instance, social smartphone apps, such as social networks, messaging services, and dating apps, could enhance their node representations using user's personal data, such as their phone's sensor data or the list of installed apps. This could significantly improve their recommendation systems and overall user experience. However, in the absence of adequate user data protection measures, this would mean that the server would need to collect raw personal data. This situation could raise privacy concerns as the collected data might be used for unauthorized purposes.

2.1.2 Challenges

Training a GNN while maintaining node attribute privacy is a complex task due to the inherent interconnectivity present in graph structures. This complexity stands in stark contrast to more conventional deep learning models where training data instances are largely independent of each other. More specifically, In GNNs, the samples aren't isolated - they are nodes within the graph, connected to each other through a web of relationships, depicted as links. During the process of training a GNN, these nodes communicate with each other through the GNN's message-passing framework, which plays a crucial role in learning the representation of the graph (Hamilton et al., 2017b). This is a unique attribute of GNNs that differentiates them from other models where data samples do not "interact" during the learning process.

This interconnectedness complicates the application of traditional privacy-preserving machine learning methods like federated learning and split learning (T. Li et al., 2019; Vepakomma et al., 2018). These paradigms rely on distributing the learning process across multiple nodes or devices while ensuring that each device's data stays on the device, thereby preserving privacy. However, in the context of a GNN, this approach incurs an excessive communication overhead due to the need to constantly exchange information between nodes. These interactions are necessary for the GNN to learn effectively but pose a significant challenge in terms of communication costs and efficiency when conventional privacy-preserving learning techniques are used. Therefore, the design of privacy-preserving mechanisms for GNNs needs to consider the relational characteristics of the graph and the high communication overhead that may occur, necessitating new strategies or modifications to existing ones. This highlights the pressing need for innovative solutions to effectively address the unique challenges presented by privacy-preserving GNN training.

2.1.3 Contributions

In this chapter, we propose a novel privacy-preserving GNN learning framework that combines the power of Graph Convolutional Networks (GCNs) (Kipf & Welling, 2017) and Local Differential Privacy (LDP) (Kasiviswanathan et al., 2011) to protect sensitive node attributes from being exposed to an honest-but-curious server (Paverd et al., 2014), who requires these features for training its own GNN (Section 2.3). By extending LDP mechanisms, the server can efficiently communicate with the graph nodes to privately collect their features and estimate the first-layer graph convolution of the GNN (Section 2.4). Our solution is optimized for more communication efficiency, and we show that it works best when the distribution of features is skewed. We derive the theoretical properties of the proposed algorithm, such as its formal privacy guarantee and error bound (Section 2.4). Finally, we conduct comprehensive experiments over several real-world graph datasets to verify our theoretical findings and demonstrate the effectiveness of our framework in privacy-preserving node classification and link prediction tasks (Section 2.6). To the best of our knowledge, this chapter defines the problem of GNN training with node attribute privacy for the first time, studying the integration of differential

privacy with graph neural networks.

2.2 Related Work

2.2.1 Graph Neural Networks

Recent years have seen a surge in applying GNNs for representation learning over graphs. A GNN consists of multiple layers, wherein each layer, the representation of a node is obtained by aggregating the previous-layer embeddings of the node and its neighbors (the neighborhood aggregation or message passing step (Hamilton et al., 2017b)), followed by a neural network transformation. Based on this idea, numerous GNN models have been proposed, including Graph Convolutional Networks (GCN) (Kipf & Welling, 2017), Graph Attention Networks (GAT) (Veličković et al., 2018), GraphSAGE (Hamilton et al., 2017a), Graph Isomorphism Networks (GIN) (K. Xu et al., 2019), Gated Graph Neural Networks (Y. Li et al., 2016), and so forth. We refer the reader to the available surveys on GNNs (Z. Wu et al., 2020; Zhou et al., 2018) for other models and their internal architecture. While there is a growing number of GNN models designed to solve different tasks in various domains, we are not aware of prior work on training GNNs in the private setting.

2.2.2 Privacy-Preserving Machine Learning

Numerous techniques based on Homomorphic Encryption (Sathya et al., 2018), Secure Multi-Party Computation (SMC) (Evans et al., 2018), Federated Learning (T. Li et al., 2019), Split Learning (Vepakomma et al., 2018), and Differential Privacy (DP) (Dwork, 2008) have been proposed to address the privacy issue when the data is sensitive and cannot be released to untrusted third-parties for model training. Homomorphic encryption allows the training and inference of a model to be performed directly on encrypted data, and has been successfully applied to deep learning models (Hesamifard et al., 2017, 2018). SMC enables two or more parties, who do not trust each other, to jointly train a model without exposing their input data to each other. It has been used for linear regression, logistic regression, and neural networks (Agrawal et al., 2019; Mohassel & Zhang, 2017; Rouhani et al., 2017). Federated and split learning algorithms allow users to keep their private data and communicate with the service provider in order to train a model, collaboratively (Bonawitz et al., 2017; Konečný et al., 2016). Finally, differential privacy provides a mathematical framework for computing statistical queries over private databases, and has been successfully adapted in deep learning systems to preserve training data privacy (S. Song et al., 2013; X. Wu et al., 2017). However, these methodologies are mostly designed for non-relational data, such as images and text. We are not aware of any prior work extending any of these techniques for GNNs.

2.3 Problem Formulation and Preliminaries

2.3.1 Problem Definition

We now define the problem of learning GNNs under the node attribute privacy setting. Assume that a server has access to a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, whose nodes and links are visible by the server, but the feature matrix $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$, comprising d-dimensional feature vectors $\mathbf{x}_v$ for each $v \in \mathcal{V}$, is private to the nodes and thus not directly accessible by the server. The question is, how can the server collaborate with the nodes to use private features and learn a GNN over $\mathcal{G}$?

As our solution is based on graph convolutional networks and local differential privacy, in the following we present the required fundamental background about these two concepts.

2.3.2 Graph Convolutional Networks

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a feature matrix $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$, an L-layer Graph Convolutional Network (GCN) (Kipf & Welling, 2017) consists of L graph convolution layers , where the embedding of a node $v \in \mathcal{V}$ at layer l is generated by aggregating the embeddings of the node's neighbors from the previous layer using the GCN graph convolution function, defined by:

$$GC^l(v) = \sum_{u \in \mathfrak{N}(v)} \frac{\mathbf{h}_u^{l-1}}{\sqrt{|\mathfrak{N}(u)||\mathfrak{N}(v)|}} \tag{2.1}$$

where $\mathfrak{N}(v)$ is the set of neighbors of node v including itself, and $\mathbf{h}_u^{l-1}$ is the embedding of node u at layer $l-1$. Also, $\mathbf{h}_v^0 = \mathbf{x_v}$, i.e. the initial representation of node v is equal to its original feature vector $\mathbf{x}_v$. The l-th layer embedding of v is then generated by $\mathbf{h}_v^l = \sigma(\mathbf{W}^l \cdot GC^l(v))$ where $\mathbf{W}^l$ is the trainable weight matrix for layer l.

The advantage of the GCN model, from a privacy-preserving perspective, is that the only point where the node features are touched is inside the graph convolution function of the very first layer. Therefore, unlike other GNN models, such as GraphSAGE or GAT, where individual node features are also used along with the aggregation of the neighboring features, GCN only requires aggregated node features – the output of the graph convolution function (2.1) in the first layer. This allows to privately calculate this aggregation using local differential privacy.

2.3.3 Local Differential Privacy

Local differential privacy (LDP) (Kasiviswanathan et al., 2011) is an increasingly used approach for collecting private data and computing statistical queries, such as mean, count, and histogram, and is already deployed by Google (Erlingsson et al., 2014), Apple (Tang et al., 2017), and Microsoft (Ding et al., 2017), for private data analytics at scale. The key idea behind LDP is that data holders do not need to share their private data with a data aggregator, but instead send a perturbed version of their data, which is meaningless individually, but can approximate

the target query when aggregated. It basically includes two steps: (i) data collection, in which each data holder perturbs its data using a special randomized mechanism $\mathscr{M}$ and sends the output to the aggregator; and (ii) estimation, in which, 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 $\mathscr{M}$ must satisfy the following definition (Kasiviswanathan et al., 2011):

Definition 1. *Given $\epsilon > 0$, a randomized mechanism $\mathscr{M}$ satisfies ϵ-local differential privacy if for all possible pairs of user's private data x and x', and for all possible outputs $y \in Range(\mathscr{M})$, we have:*

$$\Pr[\mathscr{M}(x) = y] \leq e^{\epsilon} \Pr[\mathscr{M}(x') = y] \tag{2.2}$$

The parameter ϵ in the above definition is called the *"privacy budget"* and is used to tune the utility-privacy trade-off: a smaller ϵ leads to stronger privacy guarantees, but lower utility (and vice versa). Based on the above definition, the mechanism $\mathscr{M}$ 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 he might have.

2.4 Proposed Method

Now, we describe our proposed framework for training a GNN using private node features. The key idea of our method is to approximate the first-layer graph convolution for any node $v \in \mathcal{V}$, as an aggregated statistic over the private features of v itself and its neighbors, using local differential privacy (LDP). To this end, our approach requires the first layer of the GNN used by the server (as the point of contact with the features) to be the GCN convolution layer defined by (2.1), but there is no restriction on the choice of subsequent layers (they can be GAT, GraphSAGE, etc). As a result, our framework includes two steps: (i) collecting the node features by the server through a differentially private approach, and (ii) approximating the graph convolution in the first layer using the collected features and proceeding with the training of the GNN.

There are several LDP mechanisms in the literature that are used for calculating mean value over private numeric data that can be adapted for our problem. The most well-known algorithms include Laplace Mechanism (Dwork et al., 2006), Duchi et al.'s Mechanism (Duchi et al., 2018), and Piecewise Mechanism (N. Wang et al., 2019). Here, we extend the method of Duchi for the aim of estimating the graph convolution function, which is a more complex statistic compared to the simple mean. Therefore, we call our LDP mechanism *Private Graph Convolution (PGC)*. Inspired by the work of Ding et al. (Ding et al., 2017), our mechanism is tuned for reduced communication cost between the server and the graph nodes. We derive the theoretical properties of our PGC method and discuss its differences with other mechanisms. Additional details about Laplace and Piecewise mechanisms are provided in Section 2.9.1.

Assume that each node v owns a private d-dimensional feature vector $\mathbf{x}_v = [\mathbf{x}_v^{(1)}, \mathbf{x}_v^{(1)}, \ldots, \mathbf{x}_v^{(d)}]^T$ where $\mathbf{x}_v^{(i)} \in [\alpha^{(i)}, \beta^{(i)}]$. When the feature vector of node v is needed, the server sends the parameter ϵ to v. After, the node sends to the server a d-dimensional perturbed vector $\mathbf{y}_v$ where each element $y_v^{(i)}$ is drawn independently from the following Bernoulli distribution:

$$\Pr[y_v^{(i)} \mid x_v^{(i)}] = \left(\frac{1}{e^\epsilon+1} + \frac{x_v^{(i)} - \alpha^{(i)}}{\Delta^{(i)}} \cdot \frac{e^\epsilon - 1}{e^\epsilon + 1}\right)^{y_v^{(i)}} \left(\frac{e^\epsilon}{e^\epsilon+1} + \frac{x_v^{(i)} - \alpha^{(i)}}{\Delta^{(i)}} \cdot \frac{1 - e^\epsilon}{e^\epsilon + 1}\right)^{1-y_v^{(i)}} \tag{2.3}$$

where $\Delta^{(i)} = \beta^{(i)} - \alpha^{(i)}$, and ϵ is the privacy budget parameter provided by the server. The value of $y_v^{(i)}$ is recorded by the node to be returned in subsequent calls so that the server cannot guess the node's private feature by issuing repeated queries.

Upon collecting the perturbed data, the server can estimate the first-layer graph convolution of any node v by:

$$\widehat{GC}(v) = \sum_{u \in \mathfrak{N}(v)} \frac{\mathbf{y}_v^*}{\sqrt{|\mathfrak{N}(u)||\mathfrak{N}(v)|}} \tag{2.4}$$

where $\mathbf{y}_v^*$ is the unbiased form of $\mathbf{y}_v$, defined as:

$$\mathbf{y}_v^* = \frac{(e^\epsilon+1)\mathbf{y}_v - 1}{e^\epsilon - 1} \cdot \boldsymbol{\Delta} + \boldsymbol{\alpha} \tag{2.5}$$

Here, $\boldsymbol{\alpha} = [\alpha_i]_{i=1:d}$, $\boldsymbol{\Delta} = [\Delta_i]_{i=1:d}$, and all vector multiplications and divisions are performed element-wise, thus the dot notation is abused for element-wise multiplication. Note that we can also calculate $\mathbf{y}_v^*$ at the node-side and send it to the server instead of $\mathbf{y}_v$. However, the advantage of performing this operation at the server-side is that $\mathbf{y}_v$ in this case will be binary-valued, which is much more efficient to send in terms of communication cost than the real-valued vector $\mathbf{y}_v^*$. This is also the advantage of PGC compared to the Laplace or Piecewise mechanisms, as the output of these two methods is continuous, which imposes more communication overhead compared to ours.

After this step, the server can proceed with the rest of the GNN layers to complete the training. The pseudo-code of the forward propagation of the proposed framework as an adaptation of the general GNN forward propagation (Hamilton et al., 2017b) is presented in Algorithm 1.

2.5 Theoretical Analysis

In this section, we discuss formal privacy guarantees and other theoretical properties of our method.

Lemma 1. *Algorithm 1 satisfies ϵ-local differential privacy for individual node features.*

Proof. According to (2.3), for a single output y_v corresponding to an arbitrary private input feature x_v, the probability that $y_v = 1$ ranges from $\frac{1}{e^\epsilon+1}$ to $\frac{e^\epsilon}{e^\epsilon+1}$ depending on x_v. Analogously, the probability that $y_v = 0$ also varies from $\frac{1}{e^\epsilon+1}$ to $\frac{e^\epsilon}{e^\epsilon+1}$. Therefore, for every node pairs v and u

Algorithm 1: Forward propagation for an L-Layer GNN as an adaptation of Hamilton et al. (2017b)

Input : Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$; depth L; weight matrices $\{\mathbf{W}^l, \forall l \in [1, L]\}$; non-linearity σ; differentiable aggregator functions $\{\text{AGGREGATE}_l, \forall l \in [2, L]\}$
Output: Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

Server sends $\epsilon, \boldsymbol{\alpha}, \boldsymbol{\Delta}$ to every node $v \in \mathcal{V}$.
Each node $v \in \mathcal{V}$ sends back $\mathbf{y}_v$ obtained using (2.3) to the server.

Server-side computation:
for $l = 1$ *to* L **do**
  for $v \in \mathcal{V}$ **do**
    if $l == 1$ **then**
      $\mathbf{GC}_v = \sum_{u \in \mathfrak{N}(v)} \frac{1}{\sqrt{|\mathfrak{N}(u)||\mathfrak{N}(v)|}} \left[\frac{(e^\epsilon + 1)\mathbf{y}_u - 1}{e^\epsilon - 1} \cdot \boldsymbol{\Delta} + \boldsymbol{\alpha} \right]$
      $\mathbf{h}_v^1 = \sigma\left(\mathbf{W}^1 \cdot \mathbf{GC}_v\right)$
    else
      $\mathbf{h}_{\mathfrak{N}(v)}^l \leftarrow \text{AGGREGATE}_l(\{\mathbf{h}_u^{l-1}, \forall u \in \mathfrak{N}(v)\})$
      $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^l \cdot \text{COMBINE}(\mathbf{h}_v^{l-1}, \mathbf{h}_{\mathfrak{N}(v)}^l)\right)$
    end
  end
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K$

and every output {0,1}, the ratio of the respected probabilities can be at most e^ϵ. As each nodes contributes with its feature to the server only once, due to the robustness of differentially private algorithms to post-processing (Dwork, Roth, et al., 2014), Algorithm 1 will be ϵ-LDP for individual node features. □

Lemma 2. *The graph convolution estimator* $\widehat{GC}(v)$ *defined by* (2.4) *is an unbiased estimator for the GCN graph convolution defined by* (2.1).

Proof. We need to show that $\mathbb{E}[\widehat{GC}(v)] = GC(v)$.

$$\begin{aligned}\mathbb{E}[\widehat{GC}(v)] &= \sum_{u \in \mathfrak{N}(v)} \frac{\mathbf{y}_v^*}{\sqrt{|\mathfrak{N}(u)||\mathfrak{N}(v)|}} \\ &= \sum_{u \in \mathfrak{N}(v)} \frac{1}{\sqrt{|\mathfrak{N}(u)||\mathfrak{N}(v)|}} \left[\frac{(e^\epsilon + 1)\,\mathbb{E}[\mathbf{y}_u] - 1}{e^\epsilon - 1} \cdot \boldsymbol{\Delta} + \boldsymbol{\alpha} \right] \end{aligned} \tag{2.6}$$

As $\mathbf{y}_u$ is drawn from a Bernoulli distribution, for any node u we have:

$$\mathbb{E}[\mathbf{y}_u] = \frac{1}{e^\epsilon + 1} + \frac{\mathbf{x}_u - \boldsymbol{\alpha}}{\boldsymbol{\Delta}} \cdot \frac{e^\epsilon - 1}{e^\epsilon + 1} \tag{2.7}$$

Combining (2.6) and (2.7) we get

$$\mathbb{E}[\widehat{GC}(v)] = \sum_{u \in \mathfrak{N}(v)} \frac{1}{\sqrt{|\mathfrak{N}(u)||\mathfrak{N}(v)|}} \left[\frac{(e^\epsilon + 1)(\frac{1}{e^\epsilon + 1} + \frac{\mathbf{x}_u - \boldsymbol{\alpha}}{\boldsymbol{\Delta}} \cdot \frac{e^\epsilon - 1}{e^\epsilon + 1}) - 1}{e^\epsilon - 1} \cdot \boldsymbol{\Delta} + \boldsymbol{\alpha} \right]$$

$$
\begin{aligned}
&= \sum_{u \in \mathfrak{N}(v)} \frac{1}{\sqrt{|\mathfrak{N}(u)||\mathfrak{N}(v)|}} \left[\frac{1 + \frac{\mathbf{x}_u - \boldsymbol{\alpha}}{\boldsymbol{\Delta}} (e^{\epsilon} - 1) - 1}{e^{\epsilon} - 1} \cdot \boldsymbol{\Delta} + \boldsymbol{\alpha} \right] \\
&= \sum_{u \in \mathfrak{N}(v)} \frac{1}{\sqrt{|\mathfrak{N}(u)||\mathfrak{N}(v)|}} \left[\frac{\mathbf{x}_u - \boldsymbol{\alpha}}{\boldsymbol{\Delta}} \cdot \boldsymbol{\Delta} + \boldsymbol{\alpha} \right] \\
&= \sum_{u \in \mathfrak{N}(v)} \frac{\mathbf{x}_u}{\sqrt{|\mathfrak{N}(u)||\mathfrak{N}(v)|}} \\
&= GC(v)
\end{aligned}
$$

□

Lemma 3. *With probability at least* $1 - \delta$*, for every node* v *and any arbitrary feature* $x_v \in [\alpha, \alpha + \Delta]$*, we have:*

$$
\left| \widehat{GC}(v) - GC(v) \right| \leq \Delta \cdot \frac{e^{\epsilon} + 1}{e^{\epsilon} - 1} \sqrt{\frac{1}{2} \log(\frac{2}{\delta})} \tag{2.8}
$$

Proof. Let $z_v = \frac{y_v}{\sqrt{|\mathfrak{N}(v)|}} \in [0, 1]$. Then using Hoeffding's inequality (Hoeffding, 1963), for all $t > 0$ we have:

$$
\Pr \left[\left| \sum_{u \in \mathfrak{N}(v)} z_u - \mathbb{E} \left(\sum_{u \in \mathfrak{N}(v)} z_u \right) \right| \geq t \right] \leq 2 \exp\{-\frac{2t^2}{|\mathfrak{N}(v)|}\}
$$

Substituting z_u with $\frac{y_u}{\sqrt{|\mathfrak{N}(u)|}}$ and combining with (2.7) we get

$$
\begin{aligned}
&\Pr \left[\left| \sum_{u \in \mathfrak{N}(v)} z_u - \mathbb{E} \left(\sum_{u \in \mathfrak{N}(v)} z_u \right) \right| \geq t \right] \\
&= \Pr \left[\left| \sum_{u \in \mathfrak{N}(v)} \frac{y_u}{\sqrt{|\mathfrak{N}(u)|}} - \sum_{u \in \mathfrak{N}(v)} \frac{\mathbb{E}(y_u)}{\sqrt{|\mathfrak{N}(u)|}} \right| \geq t \right] \\
&= \Pr \left[\left| \sum_{u \in \mathfrak{N}(v)} \frac{y_u}{\sqrt{|\mathfrak{N}(u)|}} - \sum_{u \in \mathfrak{N}(v)} \frac{\frac{1}{e^{\epsilon}+1} + \frac{x_u - \alpha}{\Delta} \cdot \frac{e^{\epsilon}-1}{e^{\epsilon}+1}}{\sqrt{|\mathfrak{N}(u)|}} \right| \geq t \right] \\
&= \Pr \left[\left| \sum_{u \in \mathfrak{N}(v)} \frac{1}{\sqrt{|\mathfrak{N}(u)|}} \left(y_u - \frac{1}{e^{\epsilon}+1} - \frac{x_u - \alpha}{\Delta} \cdot \frac{e^{\epsilon}-1}{e^{\epsilon}+1} \right) \right| \geq t \right] \\
&= \Pr \left[\left| \sum_{u \in \mathfrak{N}(v)} \left[\frac{1}{\sqrt{|\mathfrak{N}(u)|}} \left(\frac{(e^{\epsilon}+1) y_u - 1}{e^{\epsilon} - 1} \cdot \Delta + \alpha \right) - \frac{x_u}{\sqrt{|\mathfrak{N}(u)|}} \right] \right| \geq \frac{e^{\epsilon}+1}{e^{\epsilon}-1} \Delta \cdot t \right] \\
&= \Pr \left[\left| \sqrt{|\mathfrak{N}(v)|} \sum_{u \in \mathfrak{N}(v)} \left[\frac{1}{\sqrt{|\mathfrak{N}(u)||\mathfrak{N}(v)|}} \left(\frac{(e^{\epsilon}+1) y_u - 1}{e^{\epsilon} - 1} \cdot \Delta + \alpha \right) - \frac{x_u}{\sqrt{|\mathfrak{N}(u)||\mathfrak{N}(v)|}} \right] \right| \geq \frac{e^{\epsilon}+1}{e^{\epsilon}-1} \Delta \cdot t \right] \\
&= \Pr \left[\left| \sqrt{|\mathfrak{N}(v)|} \left(\widehat{GC}(v) - GC(v) \right) \right| \geq \frac{e^{\epsilon}+1}{e^{\epsilon}-1} \Delta \cdot t \right] \\
&= \Pr \left[\left| \widehat{GC}(v) - GC(v) \right| \geq \frac{e^{\epsilon}+1}{e^{\epsilon}-1} \frac{t}{\sqrt{|\mathfrak{N}(v)|}} \cdot \Delta \right] \leq 2 \exp\{-\frac{2t^2}{|\mathfrak{N}(v)|}\}
\end{aligned} \tag{2.9}
$$

Setting $2\exp\{-\frac{2t^2}{|\mathfrak{N}(v)|}\} = \delta$, we get $t = \sqrt{\frac{|\mathfrak{N}(v)|}{2}\log\frac{2}{\delta}}$. Then by combining with (2.9), we have

$$\Pr\left[\left|\widehat{GC}(v) - GC(v)\right| \geq \Delta \cdot \frac{e^\epsilon + 1}{e^\epsilon - 1}\sqrt{\frac{1}{2}\log(\frac{2}{\delta})}\right] \leq \delta$$

and therefore

$$\Pr\left[\left|\widehat{GC}(v) - GC(v)\right| \leq \Delta \cdot \frac{e^\epsilon + 1}{e^\epsilon - 1}\sqrt{\frac{1}{2}\log(\frac{2}{\delta})}\right] \geq 1 - \delta \tag{2.10}$$

which concludes the proof. □

Lemma 4. *For any node v and any arbitrary input feature $x_v \in [\alpha, \beta]$ with $\Delta = \beta - \alpha$, the variance of the unbiased PGC mechanism, y_v^*, is calculated by:*

$$Var[y_v^*] = \Delta^2 \frac{e^\epsilon}{(e^\epsilon - 1)^2} + (x_v - \alpha)(\beta - x_v) \tag{2.11}$$

Proof. We first calculate the variance of y_v as in (2.3). Since y_v is Bernoulli, its variance can be obtained as:

$$Var[y_v] = \mathbb{E}[y_v](1 - \mathbb{E}[y_v]) \tag{2.12}$$

Combining with (2.7), we get:

$$\begin{aligned}
Var[y_v] &= \left(\frac{1}{e^\epsilon + 1} + \frac{x_v - \alpha}{\Delta} \cdot \frac{e^\epsilon - 1}{e^\epsilon + 1}\right)\left(1 - \left[\frac{1}{e^\epsilon + 1} + \frac{x_v - \alpha}{\Delta} \cdot \frac{e^\epsilon - 1}{e^\epsilon + 1}\right]\right) \\
&= \left(\frac{1}{e^\epsilon + 1} + \frac{x_v - \alpha}{\Delta} \cdot \frac{e^\epsilon - 1}{e^\epsilon + 1}\right)\left(\frac{e^\epsilon}{e^\epsilon + 1} - \frac{x_v - \alpha}{\Delta} \cdot \frac{e^\epsilon - 1}{e^\epsilon + 1}\right) \\
&= \frac{e^\epsilon}{(e^\epsilon + 1)^2} - \frac{x_v - \alpha}{\Delta} \cdot \frac{e^\epsilon - 1}{(e^\epsilon + 1)^2} + \frac{x_v - \alpha}{\Delta} \cdot \frac{e^\epsilon(e^\epsilon - 1)}{(e^\epsilon + 1)^2} - \left(\frac{x_v - \alpha}{\Delta} \cdot \frac{e^\epsilon - 1}{e^\epsilon + 1}\right)^2 \\
&= \frac{e^\epsilon}{(e^\epsilon + 1)^2} + \left(\frac{x_v - \alpha}{\Delta} \cdot \frac{e^\epsilon - 1}{(e^\epsilon + 1)^2}\right)(e^\epsilon - 1) - \left(\frac{x_v - \alpha}{\Delta} \cdot \frac{e^\epsilon - 1}{e^\epsilon + 1}\right)^2 \\
&= \frac{e^\epsilon}{(e^\epsilon + 1)^2} + \frac{x_v - \alpha}{\Delta} \cdot \left(\frac{e^\epsilon - 1}{e^\epsilon + 1}\right)^2 - \left(\frac{x_v - \alpha}{\Delta} \cdot \frac{e^\epsilon - 1}{e^\epsilon + 1}\right)^2 \\
&= \frac{e^\epsilon}{(e^\epsilon + 1)^2} + \left(\frac{e^\epsilon - 1}{e^\epsilon + 1}\right)^2 \left[\frac{x_v - \alpha}{\Delta} - \left(\frac{x_v - \alpha}{\Delta}\right)^2\right] \\
&= \frac{e^\epsilon}{(e^\epsilon + 1)^2} + \left(\frac{e^\epsilon - 1}{e^\epsilon + 1}\right)^2 \left(\frac{x_v - \alpha}{\Delta}\right)\left(1 - \frac{x_v - \alpha}{\Delta}\right) \\
&= \frac{e^\epsilon}{(e^\epsilon + 1)^2} + \left(\frac{e^\epsilon - 1}{e^\epsilon + 1}\right)^2 \left(\frac{x_v - \alpha}{\Delta}\right)\left(\frac{\beta - x_v}{\Delta}\right)
\end{aligned} \tag{2.13}$$

Finally, using the above equation and according to (2.5), we have:

$$Var[y_v^*] = Var\left[\frac{(e^\epsilon + 1)y_v - 1}{e^\epsilon - 1} \cdot \Delta + \alpha\right]$$

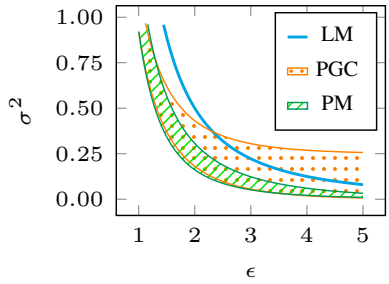


Figure 2.1: Variance of different LDP mechanisms with $\alpha = 0$, $\beta = 1$.

$$= \Delta^2 \cdot \left(\frac{e^\epsilon + 1}{e^\epsilon - 1}\right)^2 Var[y_v]$$
$$= \Delta^2 \frac{e^\epsilon}{(e^\epsilon - 1)^2} + (x_v - \alpha)(\beta - x_v) \tag{2.14}$$

□

The variance of an LDP mechanism is the key factor affecting the accuracy of the estimator, as a lower variance will usually lead to a more accurate estimation. The variance of y_v^* as shown by (2.11) depends on the original feature x_v. It can be easily shown that the variance will reaches its minimum as $x_v \to \alpha$ or $x_v \to \beta$, and gets maximized when $x_v \to \frac{\alpha+\beta}{2}$. Therefore, the *expected variance* of y_v^* is minimum when $\mathbb{E}[x_v]$ is close to either α or β, and this happens when the distribution of the features is skewed toward either the beginning or the end of their domain $[\alpha, \beta]$ (e.g. Exponential distribution).

Figure 2.1 compares the minimum and maximum variance of PGC with those of Laplace mechanism (LM) and Piecewise mechanism (PM) for an arbitrary feature x with $\alpha = 0$, $\beta = 1$, and $\epsilon \in [1, 5]$. The dotted orange area represents the region where the variance of PGC changes between its minimum and maximum, and the green hatched area corresponds to PM. LM is shown by a single blue line, as its variance does not depend on x. According to the figure, while the worst-case variance of PM is always lower than PGC and LM, this is not necessarily true in the expected case (as we will see in our experiments), especially when the feature distribution is skewed. This is because as opposed to PGC, the variance of PM is maximized at the two sides of the feature range and minimized in the middle (N. Wang et al., 2019), thus it works better if the the feature distribution is symmetric on $[\alpha, \beta]$ (e.g. Gaussian). A more detailed comparison (with LM and PM variance formulas) is available in Section 2.9.2.

2.6 Experiment Setup

We conduct extensive experiments to assess the privacy-utility trade-off of the proposed framework and evaluate its performance in two classical tasks: node classification and link prediction.

2.6.1 Datasets

In order to test the performance of the proposed method and other baselines, we use five real-world datasets with different characteristics, whose description is summarized in Table 2.1. Additional details about the datasets are as follows:

Citation networks. We use three well-known benchmark datasets, namely Cora and CiteSeer (Yang et al., 2016). These datasets are citation networks, where documents are represented as nodes, and edges represent citation links. Additionally, each node has a bag-of-words feature vector and a label indicating its category.

Elliptic. This dataset is the network of Bitcoin transactions released by Elliptic[1]. Each node in this dataset represents a transaction, and edges represent the flow of Bitcoin between two transactions. Around 23% of the nodes in the dataset have been labeled as being created by a "licit" or "illicit" entity. Vertex features comprise local and aggregated information about the transactions.

Flickr. We also use the Flickr dataset from (Zeng et al., 2019), in which nodes represent images uploaded to Flickr, and edges indicate that images share some common attributes, such as location, gallery, or comments by the same user. The dataset also contains bag-of-words feature vectors for each node, and the nodes are labeled according to their tags.

Twitch. The last dataset is collected from Twitch, which is a social networking platform for gamers. Nodes in the dataset correspond to Twitch users, and edges represent mutual friendships between them. Node features are extracted based on location, games played and liked, and streaming habits. The label of each node determines if the user uses explicit language. Here, we use the UK version of this dataset from (Rozemberczki et al., 2019).

2.6.2 Baselines

We compare the performance of the proposed method against a number of relevant baselines. Alongside our PGC method, we also use the Laplace mechanism (LM) and the Piecewise mechanism (PM) in the proposed framework for error analysis and predictive performance comparison. Additionally, to see how the performance of different LDP methods compares to the optimal case, we use standard GCN with original features without any privacy protection, denoted as RAW, whose result can be considered as an upper-bound on the privacy-preserving methods.

[1] https://www.kaggle.com/ellipticco/elliptic-data-set

Table 2.1: Descriptive statistics of the real-world datasets

Dataset	# Classes	# Nodes	# Edges	# Features	Avg. Degree	Train/ Val/Test*
CiteSeer	6	3,327	4,552	3,703	2.74	4/15/30%†
Cora	7	2,708	5,278	1,433	3.90	5/17/37%†
Elliptic	2	203,769	234,355	166	2.30	11/6/6%†
Flicker	7	89,250	449,878	500	10.08	50/25/25%
Twitch	2	7,126	35,324	2,545	9.91	50/25/25%

* Corresponds to node classification. For link prediction, all datasets use 85/5/10% splits.
† The splits do not sum to 100% as not all the nodes are labeled.

2.6.3 Experiment Settings

For the node classification task, we use standard training/validation/test splits for the citation networks as well as Flickr (Yang et al., 2016; Zeng et al., 2019). In other datasets, we randomly split labeled nodes with 50/25/25% ratios for training, validation, and test sets, respectively. For link prediction, all the datasets are randomly split with 85/5/10% ratios. We normalize the node features of all the datasets between zero and one, so in all cases, we have $\alpha = 0$ and $\Delta = 1$. We use the two-layer GCN as in (Kipf & Welling, 2017) for the node classification task. For link prediction, we use Variational Graph Auto-Encoder (Kipf & Welling, 2016) with a two-layer GCN as the encoder, with batch-normalization applied after the first convolutional layer. In all cases, we use ReLU activation function for all layers and fix the hidden and output dimensions of the two-layer GCN model (both for node classification and as the encoder of the VGAE model for link prediction) to 32 and 16, respectively. We optimize hyper-parameters for Raw method on the validation set over initial learning rate, weight decay, and dropout rate, and use the same values for all the other methods. All models are trained using Adam optimizer (Kingma & Ba, 2014) over a maximum of 500 epochs. For node classification, we train the models for a minimum of 10 epochs and stop the training if the validation loss does not decrease over 20 consecutive epochs. In the case of link prediction, we set the minimum number of epochs to 100, and check the validation loss every 10 epochs. We stop the training if the validation loss does not decrease over 10 consecutive checks. Finally, we measure the performance on the test set over 10 runs and report the average and standard deviation of the results. Implementation is done using PyTorch Geometric (Fey & Lenssen, 2019) and PyTorch Lightning (Falcon, 2019). The code to reproduce our experimental results is publicly available at https://github.com/sisaman/LPGNN/tree/dp-meet-gnn.

2.6.4 Hyper-Parameter Configuration

We performed grid search in order to find the best choices for initial learning rate, weight decay, and dropout rate based on the performance of each model, GCN for node classification

Table 2.2: Best hyper-parameter configurations based on validation set performance.

	GCN for Node Classification			VGAE for Link Prediction		
Dataset	Learning Rate	Weight Decay	Dropout	Learning Rate	Weight Decay	Dropout
CiteSeer	0.01	0.1	0.5	0.01	0.01	0
Cora	0.01	0.01	0.5	0.01	0.01	0
Elliptic	0.01	0	0	0.01	0.0001	0
Flickr	0.01	0	0	0.01	0.001	0
Twitch	0.001	0.0001	0	0.01	0.001	0

and VGAE for link prediction, across different datasets. Table 2.2 displays the best performing hyper-parameters used for our experiments.

2.7 Results and Discussion

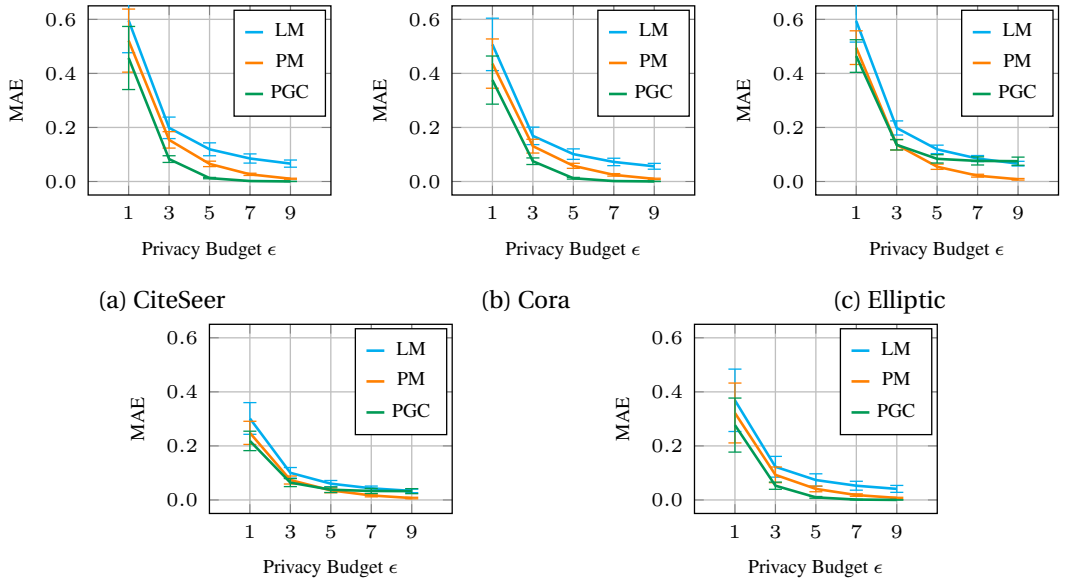
2.7.1 Analysis of Estimation Error

In the first set of experiments, we set to empirically measure the mean absolute error (MAE) in the estimation of the graph convolution for our PGC mechanism and compare it with the one generated by LM and PM. We calculate the MAE for any node v as:

$$MAE(v) = \frac{1}{d}\sum_{i=1}^{d}\left|GC(v)^{(i)} - \widehat{GC}(v)^{(i)}\right| \tag{2.15}$$

First, we look at how different LDP mechanisms perform in terms of the estimation error with respect to different values of the privacy budget ϵ. We vary the value of ϵ within $\{1,3,\ldots,9\}$, and average the MAE over all the nodes of the graph. The results for different datasets are illustrated in Figure 2.2. As expected, by increasing the value of ϵ, the MAE of all methods decreases exponentially and converges toward zero. We can see that the error of the PGC mechanism is consistently lower than other mechanisms across three out of five datasets, namely Cora, CiteSeer, and Twitch. But in the other two, Elliptic and Flickr, it falls behind PM for higher values of ϵ. This is because the features in the first three mentioned datasets are rather sparse, so the feature distribution is skewed toward zero. But in the other two datasets, the average feature value is closer to 0.5, resulting in lower error for PM. However, even in this case, when ϵ is small, PGC still results in a more accurate estimation than PM.

Next, we assess how the estimation error varies on average as a function of node degree. To this end, we set $\epsilon = 1$, and average the MAE over all the nodes v with the same degree d_v and plot the result for $d_v \in [1, d_{max}]$, where d_{max} is the 99% quantile of the degree distribution that we choose as a cut-off. The result is depicted in Figure 2.3. According to the figure, the estimation error drops first as the node degree increases but quickly converges to a constant value. This result is consistent with Lemma 3, verifying that the error is not asymptotically dependent on the node degree.



(a) CiteSeer (b) Cora (c) Elliptic

(d) Flickr (e) Twitch

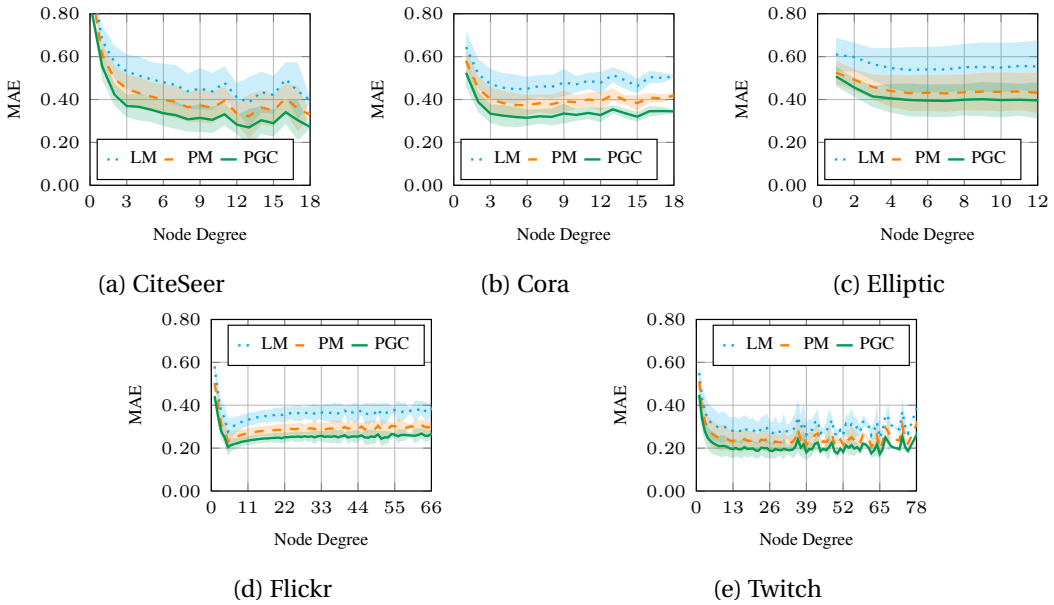
Figure 2.2: Effect of the privacy budget ϵ on the mean absolute error of the graph convolution estimation.

2.7.2 Analysis of Predictive Performance

Finally, we evaluate how our proposed framework performs in learning rich node representations for node classification and link prediction tasks under the privacy requirements. For this experiment, we selected the value of ϵ from $\{1,5,9\}$. The performance of different methods is measured in terms of micro averaged F1 score for node classification and the area under ROC curve (AUC) for link prediction. Table 2.3 presents the experimental results of different methods in both node classification and link prediction tasks. For illustrative purposes, for each value of ϵ, The closest number to the result of the RAW method (the optimal result) is bold-faced. Similar to Figure 2.2, we see that the performance of different methods (except RAW obviously) rise as we increase ϵ. The table shows that our proposed PGC method is superior to PM and LM most of the times in both node classification and link prediction across all datasets and for different values of ϵ. Analogous to the error estimation result in Figure 2.2, the exceptions here are the Elliptic and Flickr datasets, for which the PM method outperforms PGC. As mentioned previously, this is because the average feature value in these two datasets is higher than the other ones, which eventually results in lower variance, lower estimation error, and finally higher accuracy for PM.

2.7.3 Discussion on Privacy-Related Parameters

In addition to the GNN model, our framework has also its own hyper-parameters $\boldsymbol{\alpha}$, $\boldsymbol{\Delta}$, and ϵ, which are sent to the graph nodes by the server. The proposed framework requires that



(a) CiteSeer (b) Cora (c) Elliptic

(d) Flickr (e) Twitch

Figure 2.3: Mean absolute error of the graph convolution estimation w.r.t node degree with $\epsilon = 1$.

the server knows the range of private features, indicated by $\boldsymbol{\alpha}$ and $\boldsymbol{\Delta}$, which is a common assumption in the literature (Duchi et al., 2018). Technically, the server could either guess the range of the features (e.g. between 00:00 and 23:59 if the feature represents time) or could use other privacy-preserving methods, such as secure multi-party computation (Sheikh & Mishra, 2010), as a pre-processing step to determine range parameters. It is important to note that if, for any node of the graph, its requested feature lies outside the bounds provided by the server (e.g. due to inaccurate guessing), the node can simply clip its feature within the given bound.

Probably the most important parameter of the method is the privacy budget ϵ, which controls the trade-off between the accuracy of the GNN and the strength of the privacy guarantee. While there are some efforts trying to systematically determining ϵ in differentially private applications (Hsu et al., 2014; Lee & Clifton, 2011), still there is no general guideline for choosing the best value for ϵ, since the privacy guarantees enforced by differential privacy are very different based on the data domain and the target query. Therefore, previous works (Ding et al., 2017; Erlingsson et al., 2014; Tang et al., 2017) usually evaluate their systems by trying different values of ϵ, and then pick the smallest one resulting in an acceptable performance for the deployment. For instance, Microsoft uses $\epsilon = 1$ to collect telemetry data from Windows users (Ding et al., 2017), while Apple's choice of ϵ in iOS and macOS ranges between 2 and 8 for different tasks (Differential Privacy Team, n.d.). Regarding our experiments, and according to the Table 2.3, while an ϵ of 1 results in an acceptable performance in some settings (e.g. link prediction on Flickr), it seems that setting ϵ to 5 leads to near-optimal results in almost all cases.

Table 2.3: Performance of different methods in link prediction and node classification tasks.

		Node Classification (Micro-F1 %)			Link Prediction (AUC %)		
Dataset	Method	$\epsilon = 1$	$\epsilon = 5$	$\epsilon = 9$	$\epsilon = 1$	$\epsilon = 5$	$\epsilon = 9$
Citeseer	LM	37.0 ± 1.6	57.6 ± 0.9	68.5 ± 1.1	77.2 ± 0.8	81.6 ± 0.7	84.3 ± 0.9
	PM	36.8 ± 1.4	67.5 ± 1.2	$\mathbf{71.0 \pm 0.9}$	78.8 ± 1.1	84.2 ± 1.0	88.1 ± 0.6
	PGC	$\mathbf{37.3 \pm 1.3}$	$\mathbf{70.7 \pm 0.6}$	70.2 ± 0.7	$\mathbf{79.0 \pm 1.1}$	$\mathbf{87.2 \pm 0.5}$	$\mathbf{90.0 \pm 0.5}$
	Raw	69.6 ± 0.9	69.6 ± 0.9	69.6 ± 0.9	89.6 ± 0.8	89.6 ± 0.8	89.6 ± 0.8
Cora	LM	54.4 ± 0.9	73.9 ± 0.8	78.8 ± 0.7	84.9 ± 0.5	87.9 ± 0.8	89.3 ± 0.8
	PM	56.6 ± 1.9	77.8 ± 1.0	80.9 ± 0.6	85.1 ± 0.9	88.6 ± 1.0	91.0 ± 0.6
	PGC	$\mathbf{57.0 \pm 1.9}$	$\mathbf{80.2 \pm 0.6}$	$\mathbf{81.2 \pm 0.4}$	$\mathbf{85.4 \pm 0.9}$	$\mathbf{90.6 \pm 0.6}$	$\mathbf{92.1 \pm 0.5}$
	Raw	81.4 ± 0.3	81.4 ± 0.3	81.4 ± 0.3	91.9 ± 0.6	91.9 ± 0.6	91.9 ± 0.6
Elliptic	LM	$\mathbf{90.5 \pm 0.2}$	91.5 ± 0.5	92.6 ± 0.4	54.6 ± 0.3	74.5 ± 1.8	81.2 ± 0.8
	PM	$\mathbf{90.5 \pm 0.1}$	$\mathbf{92.7 \pm 0.4}$	$\mathbf{94.1 \pm 0.5}$	56.8 ± 0.7	$\mathbf{82.1 \pm 2.0}$	$\mathbf{86.6 \pm 1.1}$
	PGC	$\mathbf{90.5 \pm 0.1}$	91.8 ± 0.2	92.2 ± 0.3	$\mathbf{57.2 \pm 0.6}$	72.1 ± 1.6	73.5 ± 1.9
	Raw	95.5 ± 0.4	95.5 ± 0.4	95.5 ± 0.4	88.1 ± 0.7	88.1 ± 0.7	88.1 ± 0.7
Flickr	LM	44.4 ± 0.4	50.2 ± 0.4	51.2 ± 0.4	$\mathbf{67.7 \pm 0.7}$	72.6 ± 1.1	74.5 ± 1.6
	PM	$\mathbf{45.2 \pm 0.4}$	$\mathbf{51.3 \pm 0.3}$	$\mathbf{52.0 \pm 0.3}$	67.4 ± 0.5	$\mathbf{75.9 \pm 1.5}$	$\mathbf{77.4 \pm 1.9}$
	PGC	45.0 ± 0.5	50.9 ± 0.3	51.1 ± 0.3	$\mathbf{67.7 \pm 0.9}$	74.4 ± 1.6	75.7 ± 1.6
	Raw	52.3 ± 0.2	52.3 ± 0.2	52.3 ± 0.2	76.3 ± 2.5	76.3 ± 2.5	76.3 ± 2.5
Twitch	LM	58.7 ± 0.8	59.8 ± 0.7	60.1 ± 0.5	$\mathbf{81.1 \pm 0.3}$	82.2 ± 0.4	82.7 ± 0.5
	PM	$\mathbf{59.0 \pm 0.8}$	60.2 ± 0.8	61.3 ± 0.5	$\mathbf{81.1 \pm 0.3}$	82.5 ± 0.3	83.7 ± 0.8
	PGC	$\mathbf{59.0 \pm 0.7}$	$\mathbf{61.2 \pm 0.8}$	$\mathbf{62.5 \pm 0.2}$	$\mathbf{81.1 \pm 0.4}$	$\mathbf{83.2 \pm 0.6}$	$\mathbf{84.6 \pm 0.8}$
	Raw	62.8 ± 0.2	62.8 ± 0.2	62.8 ± 0.2	84.9 ± 1.0	84.9 ± 1.0	84.9 ± 1.0

2.8 Conclusion

In this chapter, we presented a privacy-preserving framework for graph representation learning based on graph convolutional networks and local differential privacy to address the node attribute privacy, when graph nodes have sensitive attributes that are kept private, but they could be advantageous to a GNN for learning rich node representations by a central server. To this end, we proposed *Private Graph Convolution*, which is a ϵ-locally differentially private mechanism, for graph nodes to perturb their private features before sending them to the server, and presented an unbiased estimator for the server by which it can estimate the first-layer graph convolution of the GNN using the perturbed features. Experimental experiments over real-work network datasets on node classification and link prediction tasks demonstrated that the proposed framework can manage the privacy-utility trade-off in learning representation for graph nodes, performs better than the classic Laplace mechanism consistently, and outperforms the Piecewise mechanism when ϵ is small or when the feature distribution is skewed.

Several potential future directions and improvements are imaginable for this work. One of the limitations of the current method is that for any node in the graph, a separate privacy budget is allocated for each individual feature. Therefore, the total privacy budget of a single node scales linearly with the number of features. While this is a common practice to assign separate privacy budgets to different features (for instance, Apple uses different dedicated

privacy budgets for emoji suggestions, lookup hints, QuickType suggestions, etc (Tang et al., 2017)), it can be problematic when the server request high-dimensional features which are highly correlated. Therefore, as future work, we plan to work on a multi-dimensional LDP mechanism to address this issue. As another future step, we would like to work on a hybrid mechanism that is optimal in terms of expected noise-variance. Finally, it might be interesting to address the problem of node attribute privacy using other privacy-preserving machine learning paradigms, such as secure multi-party computation. Overall, the concept of privacy-preserving graph representation learning is a novel and unexplored field of research with many potential future directions that can go beyond node attribute privacy.

2.9 Supplementary Materials

2.9.1 Related LDP Mechanisms

Laplace Mechanism

Laplace mechanism (LM) (Dwork et al., 2006) is a classic differentially private algorithm that can be used in both the global and the local differential privacy setting. In the local approach, each data subject basically adds a random noise to its data, which is drawn from $Lap(y \mid \frac{\Delta}{\epsilon})$, where:

$$Lap(y \mid \lambda) = \frac{1}{2\lambda} \exp\left(-\frac{|y|}{\lambda}\right)$$

is the zero-mean Laplace distribution with scale parameter λ. In our experiments, we independently add Laplace noise to each individual feature x_v for every node v. Formally, we set:

$$y_v^* = x_v + y_v \tag{2.16}$$

where y_v is a random noise drawn from $Lap(y_v \mid \frac{\Delta}{\epsilon})$. As the mean of this distribution is zero, the Laplace mechanism is unbiased, i.e. $\mathbb{E}[y_v^*] = x_v$. Furthermore, the variance of the Laplace mechanism is by definition equal to:

$$Var[y_v^*] = 2(\frac{\Delta}{\epsilon})^2 \tag{2.17}$$

Duchi's Mechanism

Another LDP mechanism for numeric data collection and mean estimation, from which our PGC method is derived, is proposed by Duchi et al. (Duchi et al., 2018) (denoted as DM). This mechanism accepts an input value $t \in [-1, 1]$ and outputs a perturbed value $t^* \in \{\frac{e^\epsilon+1}{e^\epsilon-1}, -\frac{e^\epsilon+1}{e^\epsilon-1}\}$ according to the following probability distribution:

$$\Pr[t^* \mid t] = \begin{cases} \frac{1}{2} + \frac{e^\epsilon-1}{2e^\epsilon+2} \cdot t, & \text{if } t^* = \frac{e^\epsilon+1}{e^\epsilon-1} \\ \frac{1}{2} - \frac{e^\epsilon-1}{2e^\epsilon+2} \cdot t, & \text{if } t^* = -\frac{e^\epsilon+1}{e^\epsilon-1} \end{cases} \tag{2.18}$$

Duchi et al. show that the above mechanism satisfies ϵ-LDP, and that t^* is an unbiased estimator of the input t. Also, the variance of the mechanism is calculated as:

$$Var[t^*] = \left(\frac{e^\epsilon + 1}{e^\epsilon - 1}\right)^2 - t^2 \tag{2.19}$$

Derivation of PGC from Duchi's mechanism. Here we describe how the PGC method is derived from DM. First, we modify DM to accept inputs withing the range $[\alpha, \beta]$ for any individual feature x_v of any node v:

$$\begin{aligned}
\alpha \le x_v \le \beta &\Longrightarrow 0 \le x_v - \alpha \le \Delta \\
&\Longrightarrow 0 \le \frac{x_v - \alpha}{\Delta} \le 1 \\
&\Longrightarrow 0 \le \frac{2(x_v - \alpha)}{\Delta} \le 2 \\
&\Longrightarrow -1 \le \frac{2(x_v - \alpha)}{\Delta} - 1 \le 1
\end{aligned} \tag{2.20}$$

Based on the above, let t_v be defined as:

$$t_v = \frac{2(x_v - \alpha)}{\Delta} - 1 \tag{2.21}$$

which is in the range $[-1, 1]$ and thus can be used as the input of DM. From (2.18), we have:

$$\Pr\left[t_v^* = \frac{e^\epsilon + 1}{e^\epsilon - 1} \mid t_v\right] = \frac{1}{2} + \frac{e^\epsilon - 1}{2e^\epsilon + 2} \cdot t_v$$

Substituting (2.21) in the above, we get:

$$\begin{aligned}
\Pr\left[t_v^* = \frac{e^\epsilon + 1}{e^\epsilon - 1} \mid x_v\right] &= \frac{1}{2} + \frac{e^\epsilon - 1}{2e^\epsilon + 2} \cdot \left(\frac{2(x_v - \alpha)}{\Delta} - 1\right) \\
&= \frac{1}{2} + \frac{x_v - \alpha}{\Delta} \cdot \frac{e^\epsilon - 1}{e^\epsilon + 1} - \frac{1}{2} \cdot \frac{e^\epsilon - 1}{e^\epsilon + 1} \\
&= \frac{1}{2}\left(1 - \frac{e^\epsilon - 1}{e^\epsilon + 1}\right) + \frac{x_v - \alpha}{\Delta} \cdot \frac{e^\epsilon - 1}{e^\epsilon + 1} \\
&= \frac{1}{2} \cdot \frac{2}{e^\epsilon + 1} + \frac{x_v - \alpha}{\Delta} \cdot \frac{e^\epsilon - 1}{e^\epsilon + 1} \\
&= \frac{1}{e^\epsilon + 1} + \frac{x_v - \alpha}{\Delta} \cdot \frac{e^\epsilon - 1}{e^\epsilon + 1}
\end{aligned} \tag{2.22}$$

Similar to the above, we will have:

$$\Pr\left[t_v^* = -\frac{e^\epsilon + 1}{e^\epsilon - 1} \mid x_v\right] = \frac{e^\epsilon}{e^\epsilon + 1} - \frac{x_v - \alpha}{\Delta} \cdot \frac{e^\epsilon - 1}{e^\epsilon + 1} \tag{2.23}$$

Analogous to (Ding et al., 2017) and to achieve communication efficiency, for each node v, we define $y_v \in \{0,1\}$ to be the output of the data collection step, such that:

$$y_v = \begin{cases} 1, & \text{if } t_v^* = \frac{e^\epsilon+1}{e^\epsilon-1} \\ 0, & \text{if } t_v^* = -\frac{e^\epsilon+1}{e^\epsilon-1} \end{cases} \tag{2.24}$$

Combining (2.22), (2.23), and (2.24), we get:

$$\Pr[y_v|x_v] = \begin{cases} \frac{1}{e^\epsilon+1} + \frac{x_v-\alpha}{\Delta} \cdot \frac{e^\epsilon-1}{e^\epsilon+1}, & \text{if } y_v = 1 \\ \frac{e^\epsilon}{e^\epsilon+1} - \frac{x_v-\alpha}{\Delta} \cdot \frac{e^\epsilon-1}{e^\epsilon+1}, & \text{if } y_v = 0 \end{cases} \tag{2.25}$$

which is equal to PGC data collection mechanism defined in (2.3). From the above, we have:

$$\mathbb{E}[y_v] = \frac{1}{e^\epsilon+1} + \frac{x_v-\alpha}{\Delta} \cdot \frac{e^\epsilon-1}{e^\epsilon+1} \tag{2.26}$$

which is clearly biased as the expectation is not equal to x_v. To get an unbiased estimator, let y_v^* be a random variable, such that $\mathbb{E}[y_v^*] = x_v$. Therefore, Combining with the above equation, we have:

$$\begin{aligned} \frac{1}{e^\epsilon+1} + \frac{\mathbb{E}[y_v^*]-\alpha}{\Delta} \cdot \frac{e^\epsilon-1}{e^\epsilon+1} &= \mathbb{E}[y_v] \\ 1 + \frac{\mathbb{E}[y_v^*]-\alpha}{\Delta} \cdot (e^\epsilon-1) &= (e^\epsilon+1)\,\mathbb{E}[y_v] \\ \frac{\mathbb{E}[y_v^*]-\alpha}{\Delta} &= \frac{(e^\epsilon+1)\,\mathbb{E}[y_v]-1}{e^\epsilon-1} \\ \mathbb{E}[y_v^*] &= \frac{(e^\epsilon+1)\,\mathbb{E}[y_v]-1}{e^\epsilon-1}\Delta + \alpha \end{aligned}$$

Finally, removing the expectations from both sides, we end up with:

$$y_v^* = \frac{(e^\epsilon+1)\,y_v-1}{e^\epsilon-1}\Delta + \alpha \tag{2.27}$$

which is the scalar form of the unbiased estimator defined by (2.5).

Piecewise Mechanism

A recently proposed LDP algorithm for numeric mean estimation is the Piecewise mechanism (PM) by Wang et al. (N. Wang et al., 2019). This mechanism assumes that the input t is again

in the range $[-1,1]$ and returns a perturbed value $t^* \in [-C,C]$, where:

$$C = \frac{\exp(\epsilon/2)+1}{\exp(\epsilon/2)-1} \tag{2.28}$$

Then, the output t^* is drawn randomly from the following piecewise constant distribution:

$$\Pr(t^* \mid t) = \begin{cases} p, & \text{if } t^* \in [\ell(t), r(t)], \\ \frac{p}{\exp(\epsilon)}, & \text{if } t^* \in [-C, \ell(t)) \cup (r(t), C]. \end{cases} \tag{2.29}$$

where

$$p = \frac{\exp(\epsilon)-\exp(\epsilon/2)}{2\exp(\epsilon/2)+2},$$
$$\ell(t) = \frac{C+1}{2}\cdot t - \frac{C-1}{2}, \text{ and}$$
$$r(t) = \ell(t) + C - 1.$$

Wang et al. prove that their mechanism is unbiased ($\mathbb{E}[t^*] = t$), and the variance is calculated as:

$$Var[t^*] = \frac{t^2}{e^{\epsilon/2}-1} + \frac{e^{\epsilon/2}+3}{3(e^{\epsilon/2}-1)^2} \tag{2.30}$$

Integration of PM into the proposed framework. In order to make the variance of PM comparable to PGC's defined in (2.11), we extend the mechanism to accept inputs withing the range $[\alpha,\beta]$ for any individual feature x_v of any node v. Based on (2.20), we set:

$$\begin{aligned} t_v &= \frac{2(x_v-\alpha)}{\Delta} - 1 \\ &= \frac{2(x_v-\alpha)-(\beta-\alpha)}{\Delta} \\ &= \frac{2}{\Delta}(x_v - \frac{\alpha+\beta}{2}) \end{aligned} \tag{2.31}$$

Combining (2.30) and (2.31), the variance becomes:

$$\begin{aligned} Var[t_v^*] &= \frac{{t_v}^2}{e^{\epsilon/2}-1} + \frac{e^{\epsilon/2}+3}{3(e^{\epsilon/2}-1)^2} \\ &= \frac{\left(\frac{2}{\Delta}(x_v - \frac{\alpha+\beta}{2})\right)^2}{e^{\epsilon/2}-1} + \frac{e^{\epsilon/2}+3}{3(e^{\epsilon/2}-1)^2} \\ &= \frac{4}{\Delta^2}\frac{\left(x_v - \frac{\alpha+\beta}{2}\right)^2}{e^{\epsilon/2}-1} + \frac{e^{\epsilon/2}+3}{3(e^{\epsilon/2}-1)^2} \end{aligned} \tag{2.32}$$

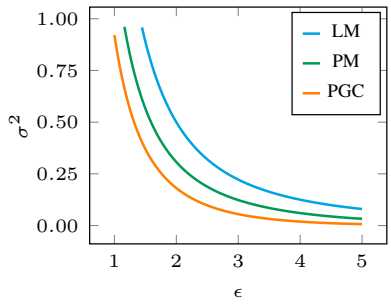


Figure 2.4: Variance of different LDP mechanisms for $x_v = \alpha$ (or $x_v = \beta$). The values of α and β was set to 0 and 1, respectively.

Let y_v^* be a random variable such that $\mathbb{E}[y_v^*] = x_v$. Since $\mathbb{E}[t_v^*] = t_v$, using (2.31) we have:

$$\begin{aligned}\mathbb{E}[t_v^*] &= \frac{2}{\Delta}(\mathbb{E}[y_v^*] - \frac{\alpha+\beta}{2}) \\ &= \mathbb{E}\left[\frac{2}{\Delta}(y_v^* - \frac{\alpha+\beta}{2})\right]\end{aligned}$$

removing the expectations from both sides and rearranging, we get:

$$y_v^* = \frac{\Delta}{2}(t_v^* + 1) + \alpha \tag{2.33}$$

The variance of y_v^* is then derived as:

$$Var[y_v^*] = Var\left[\frac{\Delta}{2}(t_v^* + 1) + \alpha\right] = \frac{\Delta^2}{4} Var[t_v^*]$$

Finally, combining with (2.32), we have:

$$Var[y_v^*] = \frac{\left(x_v - \frac{\alpha+\beta}{2}\right)^2}{e^{\epsilon/2} - 1} + \frac{e^{\epsilon/2} + 3}{3(e^{\epsilon/2} - 1)^2} \cdot \frac{\Delta^2}{4} \tag{2.34}$$

2.9.2 Variance Comparison of LDP Mechanisms

Equation (2.34) clearly shows that the minimum variance of PM is achieved when $x_v = \frac{\alpha+\beta}{2}$, while its variance is maximized when $x_v = \alpha$ or $x_v = \beta$. In other words, as opposed to the PGC mechanism, whose variance decreases to its minimum as x_v approaches α or β, the variance of PM gets elevated toward its maximum. This is illustrated further in Figure 2.4, where the variance of the three LDP mechanisms, LM, PM, and PGC, is depicted for the case when $x_v = \alpha$ or $x_v = \beta$. It can be observed that in this case, PGC generates less variance than both LM and PM, regardless of the value of ϵ. Therefore, considering that the feature x_v is generated from some prior distribution $p(x)$ defined on $[\alpha, \beta]$, PGC is expected to be a better choice if $p(x)$ is skewed toward either α or β, e.g. when $p(x)$ is a truncated exponential distribution defined

Table 2.4: Average feature value $\bar{x}$ in different datasets

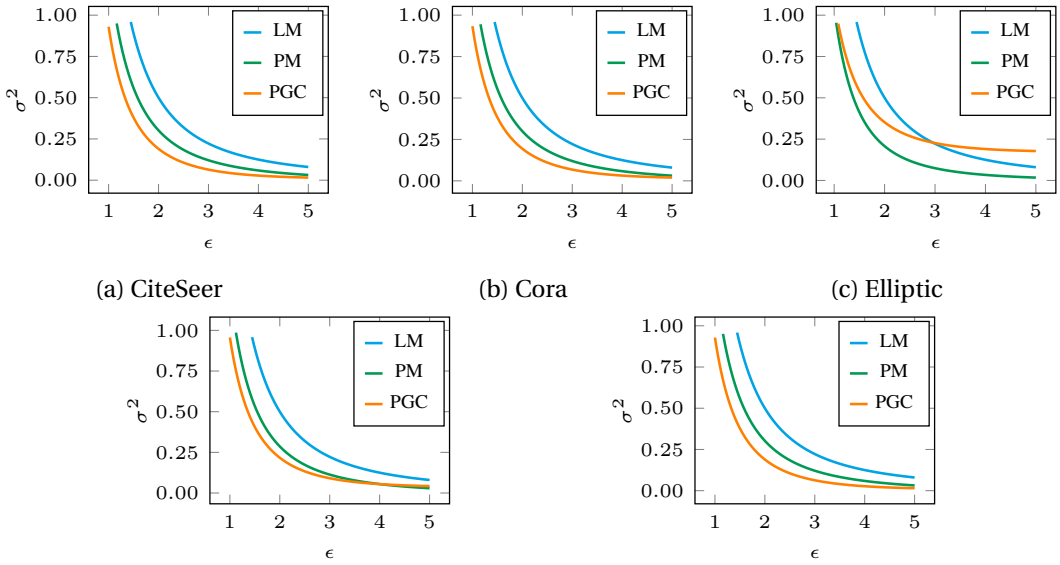
DATASET	CITESEER	CORA	ELLIPTIC	FLICKR	TWITCH
$\bar{x}$	0.009	0.013	0.219	0.037	0.008

over the range $[\alpha, \beta]$, so that $\mathbb{E}[x]$ is close to α or β. If α is zero, this implies that PGC works best if most of the features are sparse. On the contrary, if $\mathbb{E}[x]$ is close to the middle of the range $[\alpha, \beta]$, e.g. when $p(x)$ is a Gaussian or uniform distribution over $[\alpha, \beta]$, then PM is preferred over PGC (as an extension of DM) when ϵ is greater than around 0.61 (N. Wang et al., 2019).

Figure 2.5 exhibits the empirical variance of the three LDP methods for $\epsilon \in [1, 5]$ across different datasets. For each dataset, the value of x_v in the PM and PGC's variance formulas (equations (2.34) and (2.11)) is replaced by the average of all the features over all the nodes, denoted as $\bar{x}$:

$$\bar{x} = \frac{1}{d \cdot |\mathcal{V}|} \sum_{v \in \mathcal{V}} \sum_{i=1}^{d} x_v^{(i)} \tag{2.35}$$

Here, we normalized the features between 0 and 1, so that they have the same scale. The value of $\bar{x}$ for different datasets is shown in Table 2.4. According to Figure 2.5 and Table 2.4, it can be observed that for all the datasets except Elliptic, the empirical variance of PGC is lower than PM and LM, as in these datasets the average feature value tends to be very close to 0. However, in case of Elliptic, for which $\bar{x}$ is far greater than the rest of the datasets, the variance of PGC exceeds PM, and for higher values of ϵ it also passes LM. This result is completely in line with error estimation, node classification, and link prediction results presented in Section 2.6.



(a) CiteSeer (b) Cora (c) Elliptic

(d) Flickr (e) Twitch

Figure 2.5: Variance of LDP mechanisms based on different values of the privacy budget ϵ obtained using empirical average of features across different datasets.

3 Locally Private Graph Neural Networks

In the preceding chapter, we introduced the problem of node attribute privacy and presented a privacy-preserving protocol based on Local Differential Privacy (LDP) to learn Graph Convolutional Networks over graphs with locally private attributes while the structure remains publicly accessible. In this chapter, we broaden our scope from merely focusing on node attribute privacy to encompassing node data privacy. This expanded problem definition incorporates not just the attributes of nodes but also their labels as private information, while the relational information - the graph edges - still remains publicly available to the central server for the training of a Graph Neural Network (GNN).

To tackle this more comprehensive privacy concern, we introduce an architecture-agnostic, privacy-preserving GNN learning framework, termed LPGNN, which confers formal privacy guarantees grounded in LDP principles. Specifically, we develop a locally private mechanism to perturb and compress node features, which the server can efficiently collect to approximate the GNN's neighborhood aggregation step. Furthermore, to improve the accuracy of the estimation, we prepend to the GNN a denoising layer, called KProp, which is based on the multi-hop aggregation of node features. Finally, we propose a robust algorithm for learning with privatized noisy labels, where we again benefit from KProp's denoising capability to increase the accuracy of label inference for node classification. This chapter will explore these concepts in detail, providing a comprehensive understanding of the LPGNN framework and its implications for privacy-preserving machine learning on graphs.

The rest of this chapter is organized as follows. Section 3.1 provides a detailed introduction to the problem of node data privacy, its challenges, and our contributions. Section 3.2 reviews related works, and Section 3.3 formally defines the problem and provides the necessary backgrounds. Then, in Section 3.4, we explain our locally private GNN training algorithm. Details of experimental setup and results are explained in Section 3.5 and Section 3.6, respectively. We discuss the proposed method in Section 3.7, and finally, we conclude the chapter in Section 3.8.

The work presented in this chapter was originally published in:

- Sajadmanesh, S., & Gatica-Perez, D. (2021). Locally private graph neural networks. *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2130–2145

3.1 Introduction

3.1.1 Problem and Motivation

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) (Scarselli et al., 2008). GNNs have shown superior performance in a wide range of applications in social sciences (Hamilton et al., 2017a), biology (Rhee et al., 2017), molecular chemistry (Duvenaud et al., 2015), and so on, achieving state-of-the-art results in various graph-based learning tasks, such as node classification (Kipf & Welling, 2017), link prediction (M. Zhang & Chen, 2018), and community detection (Z. Chen et al., 2017). 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.

In light of these privacy constraints, we define the problem of node data privacy. As illustrated in Figure 3.1, 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.

3.1.2 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

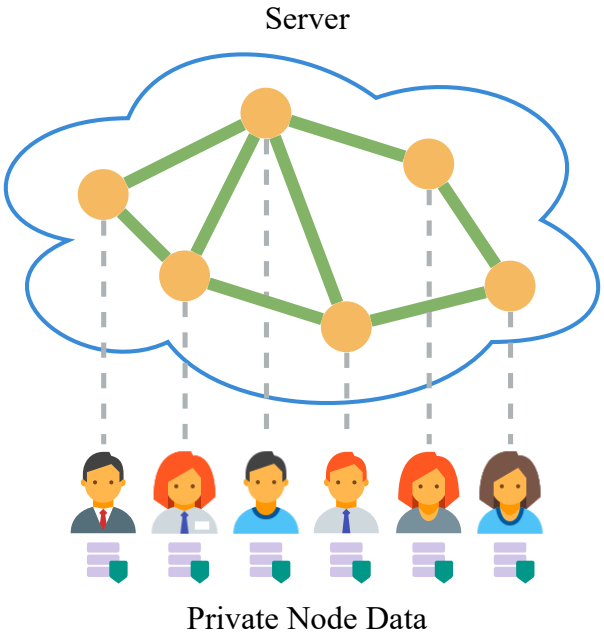


Figure 3.1: The node data privacy problem. A cloud server (e.g., a social network server) has a graph (e.g., the social graph), whose nodes, which may correspond to real users, have some private data that the server wishes to utilize for training a GNN on the graph, but cannot simply collect them due to privacy constraints.

exchange information through the message-passing framework during training (Hamilton et al., 2017b). This fact renders common collaborative learning paradigms, such as federated learning (Kairouz et al., 2019), 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 this chapter, 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.

Apart from that, our problem is also different from and more challenging than differentially private model training, where the data is completely accessible by the server, and the aim is to train and then publish a model, either entirely or as a service, in a way that the released model or its predictions do not expose information about the training data. This is typically done by adding differentially private noise to the gradients of the model during training (Abadi et al., 2016), or training the model using a noisy loss function (Zhao et al., 2019). In our problem setting, however, the server is not trusted and thus does not have direct access to the data (node features), and accordingly, an extra layer of protection should be implemented between the user's data and the server.

3.1.3 Contributions

In this chapter, 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) (Kasiviswanathan et al., 2011), 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 (Kairouz et al., 2016). 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.

Overall, our main contributions are summarized as follows:

- We introduce an LDP mechanism to address the high dimensional feature perturbation that is also very efficient in terms of communication cost.
- We propose an effective graph convolution layer, called KProp, that is attached before the backbone GNN to denoise the GNN's input by aggregating node features from multi-hop neighbors.
- We propose an end-to-end approach to train our locally private GNN with private node labels that does not rely on any clean data (features or labels), neither for training nor

validation and hyper-parameter optimization.

- We discuss why our LDP-based approach is much more efficient than other solutions based on federated learning, split learning, and secure multi-party computation.

- We conduct extensive experiments which show that our method performs very well in terms of accuracy-privacy trade-off.

3.2 Related Work

3.2.1 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 (Kipf & Welling, 2017), Graph Attention Networks (Veličković et al., 2018), GraphSAGE (Hamilton et al., 2017a), Graph Isomorphism Networks (K. Xu et al., 2019), Jumping Knowledge Networks (K. Xu et al., 2018), Gated Graph Neural Networks (Y. Li et al., 2016), and so on. We refer the reader to the available surveys on GNNs (Hamilton et al., 2017b; Z. Wu et al., 2020) for other models and discussion on their performance and applications.

3.2.2 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 (Bassily & Smith, 2015; Erlingsson et al., 2014; T. Wang et al., 2017), mean (Ding et al., 2017; Duchi et al., 2018; N. Wang et al., 2019) heavy hitter (T. Wang et al., 2019), and frequent itemset mining (Qin et al., 2016). There are also some works focusing on learning problems, such as probability distribution estimation (Acharya et al., 2018; Duchi et al., 2018; Kairouz et al., 2016), heavy hitter discovery (Bassily et al., 2017; Bun et al., 2019; T. Wang et al., 2019), frequent new term discovery (N. Wang et al., 2018), marginal release (Cormode et al., 2018), clustering (Nissim & Stemmer, 2018), and hypothesis testing (Gaboardi & Rogers, 2018). Specifically, LDP frequency oracles are considered as fundamental primitives in LDP, and numerous mechanisms have been proposed (Acharya et al., 2019; Bassily & Smith, 2015; Bassily et al., 2017; Erlingsson et al., 2014; T. Wang et al., 2017; Ye & Barg, 2018). Most works rely on techniques like Hadamard transform (Acharya et al., 2019; Bassily et al., 2017) and hashing (T. Wang et al., 2017). LDP frequency oracles are also used in other tasks, e.g., frequent itemset mining (Qin et al., 2016; T. Wang et al., 2018), and histogram estimation (Kairouz et al., 2016; S. Wang et al., 2016; Y. Wang et al., 2016).

3.2.3 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. X. He, Jia, et al. (2021b) 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. (2020) 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, B. Wu et al. (2020) 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.

3.2.4 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. D. Xu et al. (2018) proposed a differentially private graph embedding method by applying the objective perturbation on the loss function of matrix factorization. S. Zhang and Ni (2019) proposed a differentially private perturbed gradient descent method based on Lipschitz condition (Jha & Raskhodnikova, 2013) for matrix factorization-based graph embedding. Both of these methods target classic graph embedding algorithms and not GNNs. K. Li et al. (2020) presented a graph adversarial training framework that integrates disentangling and purging mechanisms to remove users' private information from learned node representations. Liao et al. (2020) 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. (2019) proposed a GNN based on structural similarity and federated learning to hide content and structure information. Zhou, Chen, et al. (2020) 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, M. Jiang et al. (2020) 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.

3.3 Preliminaries

3.3.1 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 chapter.

3.3.2 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:

$$\mathbf{h}_{\mathfrak{N}_v}^l = \mathtt{agg}_l\left(\{\mathbf{h}_u^{l-1}, \forall u \in \mathfrak{N}_v\}\right) \tag{3.1}$$

$$\mathbf{h}_v^l = \mathtt{upd}_l\left(\mathbf{h}_{\mathfrak{N}_v}^l\right) \tag{3.2}$$

where $\mathfrak{N}_v$ 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 $l-1$. $\mathtt{agg}_l(.)$ and $\mathbf{h}_{\mathfrak{N}_v}^l$ are respectively the l-th layer differentiable, permutation invariant aggregator function (such as mean, sum, or max) and its output on $\mathfrak{N}_v$. Finally, $\mathtt{upd}_l(.)$ 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.

3.3.3 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 (Erlingsson et al., 2014), Apple (Thakurta et al., 2017), and Microsoft (Ding et al., 2017). 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 $\mathscr{M}$ 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 $\mathscr{M}$ must satisfy the following definition (Kasiviswanathan et al., 2011):

Definition 2. *Given $\epsilon > 0$, a randomized mechanism $\mathscr{M}$ satisfies ϵ-local differential privacy, if for all possible pairs of user's private data x and x', and for all possible outputs $y \in Range(\mathscr{M})$, we have:*

$$\Pr[\mathscr{M}(x) = y] \leq e^{\epsilon} \Pr[\mathscr{M}(x') = y] \tag{3.3}$$

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 $\mathscr{M}$ 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.

3.4 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 agg 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 agg function average out the injected noise (to an extent, not entirely), yielding a relatively good approximation of the neighborhood aggregation for the subsequent upd 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 (Dwork, Roth, et al., 2014)), 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 (Ding et al., 2017) 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 (Kairouz et al., 2016), 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 (Y. Li, Chen, et al., 2021; NT et al., 2019; Patrini et al., 2017; H. Song et al., 2020; Yi & Wu, 2019; H. Zhang et al., 2017; Z. Zhang & Sabuncu, 2018), almost all the previous works either need clean

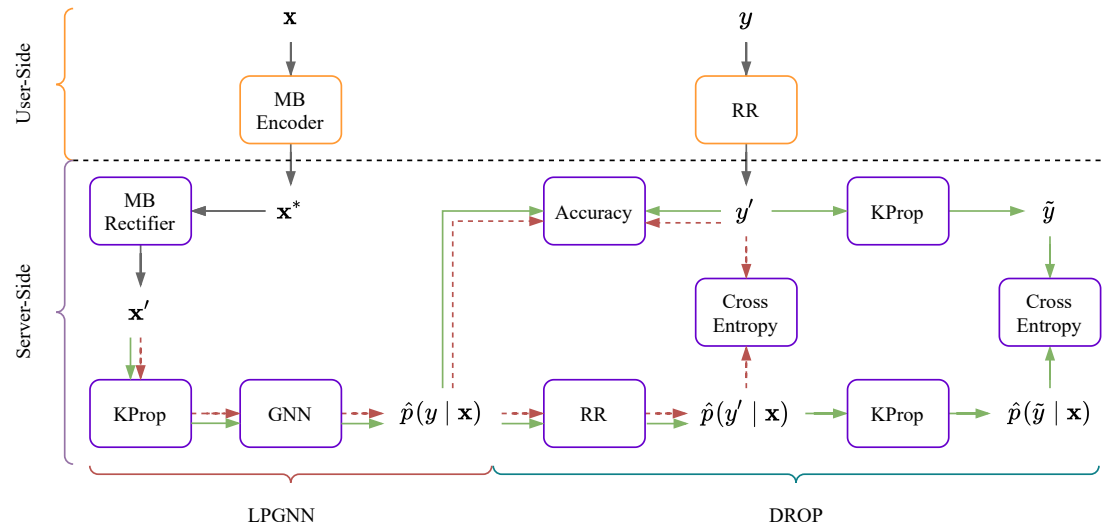


Figure 3.2: Overview of our locally private GNN training framework, featuring the multi-bit mechanism (MB Encoder and MB Rectifier), randomized response (RR), KProp layers, and Drop training. Users run multi-bit encoder and randomized response on their private features and labels, respectively, and send the output to the server, after which training begins. Green solid arrows and red dashed arrows indicate the training and validation paths, respectively.

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 (H. Wang & Leskovec, 2020), 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 3.2. 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.

Algorithm 2: 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 $\mathscr{S}$ 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 $i \in \mathscr{S}$ 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 $\mathbf{x}^* = [x_1^*, \ldots, x_d^*]^T$

3.4.1 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 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 (Ding et al., 2017), 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 agg function.

Algorithm 2 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.

Theorem 1. *The multi-bit encoder presented in Algorithm 2 satisfies ϵ-local differential privacy for each node.*

Proof. Let $\mathscr{M}(\mathbf{x})$ denote the multi-bit encoder (Algorithm 2) applied on the input vector $\mathbf{x}$. Let $\mathbf{x}^* = \mathscr{M}(\mathbf{x})$ 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 $\frac{\Pr[\mathscr{M}(\mathbf{x}_1)=\mathbf{x}^*]}{\Pr[\mathscr{M}(\mathbf{x}_2)=\mathbf{x}^*]} \leq e^{\epsilon}$.

According to Algorithm 2, 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 $i \notin \mathscr{S}$ with probability $1 - \frac{m}{d}$, therefore:

$$\frac{\Pr[\mathscr{M}(\mathbf{x}_1)_i = 0]}{\Pr[\mathscr{M}(\mathbf{x}_2)_i = 0]} = \frac{1 - m/d}{1 - m/d} = 1 \leq e^{\epsilon}, \quad \forall \epsilon > 0 \tag{3.4}$$

According to Algorithm 2, 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:

$$\begin{aligned}\frac{\Pr[\mathscr{M}(\mathbf{x}_1)_i \in \{-1,1\}]}{\Pr[\mathscr{M}(\mathbf{x}_2)_i \in \{-1,1\}]} &\leq \frac{\max \Pr[\mathscr{M}(\mathbf{x}_1)_i \in \{-1,1\}]}{\min \Pr[\mathscr{M}(\mathbf{x}_2)_i \in \{-1,1\}]} \\ &\leq \frac{\frac{m}{d} \cdot \frac{e^{\epsilon/m}}{e^{\epsilon/m}+1}}{\frac{m}{d} \cdot \frac{1}{e^{\epsilon/m}+1}} \leq e^{\epsilon/m}\end{aligned} \tag{3.5}$$

Consequently, we have:

$$\begin{aligned}\frac{\Pr[\mathscr{M}(\mathbf{x}_1) = \mathbf{x}^*]}{\Pr[\mathscr{M}(\mathbf{x}_2) = \mathbf{x}^*]} &= \prod_{i=1}^{d} \frac{\Pr\left[\mathscr{M}(x_1)_i = x_i^*\right]}{\Pr\left[\mathscr{M}(x_2)_i = x_i^*\right]} \\ &= \prod_{j|x_j^*=0} \frac{\Pr\left[\mathscr{M}(x_1)_j = 0\right]}{\Pr\left[\mathscr{M}(x_2)_j = 0\right]} \\ &\quad \times \prod_{k|x_k^* \in \{-1,1\}} \frac{\Pr[\mathscr{M}(x_1)_k \in \{-1,1\}]}{\Pr[\mathscr{M}(x_2)_k \in \{-1,1\}]} && (3.6) \\ &= \prod_{x_k^* \in \{-1,1\}} \frac{\Pr[\mathscr{M}(x_1)_k \in \{-1,1\}]}{\Pr[\mathscr{M}(x_2)_k \in \{-1,1\}]} && (3.7) \\ &\leq \prod_{x_k^* \in \{-1,1\}} e^{\epsilon/m} && (3.8) \\ &\leq e^{\epsilon} && (3.9)\end{aligned}$$

which concludes the proof. In the above, (3.7) and (3.8) follows from applying (3.4) and (3.5), respectively, and (3.9) follows from the fact that exactly m number of input features result in non-zero output.

□

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:

$$\mathbf{x}' = Rect(\mathbf{x}^*) = \frac{d(\beta-\alpha)}{2m} \cdot \frac{e^{\epsilon/m}+1}{e^{\epsilon/m}-1} \cdot \mathbf{x}^* + \frac{\alpha+\beta}{2} \tag{3.10}$$

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 1. *The multi-bit rectifier defined by* (3.10) *is unbiased.*

We first establish the following lemma and then prove Proposition 1:

Lemma 5. *Let* $\mathbf{x}^*$ *be the output of Algorithm 2 on the input vector* $\mathbf{x}$*. For any dimension* $i \in \{1,2,\ldots,d\}$*, we have:*

$$\mathbb{E}[x_i^*] = \frac{m}{d} \cdot \frac{e^{\epsilon/m}-1}{e^{\epsilon/m}+1} \cdot \left(2 \cdot \frac{x_i-\alpha}{\beta-\alpha} - 1\right) \tag{3.11}$$

and

$$Var[x_i^*] = \frac{m}{d} - \left[\frac{m}{d} \cdot \frac{e^{\epsilon/m}-1}{e^{\epsilon/m}+1} \cdot \left(2 \cdot \frac{x_i-\alpha}{\beta-\alpha} - 1\right)\right]^2 \tag{3.12}$$

Proof. For the expectation, we have:

$$\begin{aligned}\mathbb{E}[x_i^*] &= \mathbb{E}[x_i^* \mid s_i = 0]\Pr(s_i = 0) + \mathbb{E}[x_i^* \mid s_i = 1]\Pr(s_i = 1) \\ &= \frac{m}{d} \cdot (2\mathbb{E}[t_i] - 1)\end{aligned} \tag{3.13}$$

Since t_i is a Bernoulli random variable, we have:

$$\mathbb{E}[t_i] = \frac{1}{e^{\epsilon/m}+1} + \frac{x_i-\alpha}{\beta-\alpha} \cdot \frac{e^{\epsilon/m}-1}{e^{\epsilon/m}+1} \tag{3.14}$$

Combining (3.13) and (3.14) yields:

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

$$= \frac{m}{d} \cdot \frac{e^{\epsilon/m}-1}{e^{\epsilon/m}+1} \cdot \left(2 \cdot \frac{x_i - \alpha}{\beta - \alpha} - 1\right) \tag{3.15}$$

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 (3.15), we get:

$$Var\left[x_i^*\right] = \frac{m}{d} - \left[\frac{m}{d} \cdot \frac{e^{\epsilon/m}-1}{e^{\epsilon/m}+1} \cdot \left(2 \cdot \frac{x_i - \alpha}{\beta - \alpha} - 1\right)\right]^2 \tag{3.16}$$

□

Now we prove Proposition 1.

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 $i \in \{1, 2, \ldots, d\}$.

$$\mathbb{E}\left[x'_{v,i}\right] = \frac{d(\beta-\alpha)}{2m} \cdot \frac{e^{\epsilon/m}+1}{e^{\epsilon/m}-1} \cdot \mathbb{E}\left[x^*_{v,i}\right] + \frac{\alpha+\beta}{2} \tag{3.17}$$

Applying Lemma 5 yields:

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

□

Proposition 2. *For any node v and any $i \in \{1, 2, \ldots, d\}$, the variance of the multi-bit rectifier defined by (3.10) at dimension i is:*

$$Var[x'_{v,i}] = \frac{d}{m} \cdot \left(\frac{\beta-\alpha}{2} \cdot \frac{e^{\epsilon/m}+1}{e^{\epsilon/m}-1}\right)^2 - \left(x_{v,i} - \frac{\alpha+\beta}{2}\right)^2 \tag{3.18}$$

Proof. According to (3.10), 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 5 yields:

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

□

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 2 to find the optimal sampling parameter m in the multi-bit encoder (Algorithm 2) that minimizes the rectifier's variance, as follows:

Proposition 3. *The optimal value of the sampling parameter m in Algorithm 2, denoted by $m^\star$, is obtained as:*

$$m^\star = \max(1, \min(d, \left\lfloor\frac{\epsilon}{2.18}\right\rfloor)) \tag{3.19}$$

Proof. We look for a value of m that minimizes the variance of the multi-bit rectifier defined by (3.10), 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 2, $Var[x'_{v,i}]$ 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:

$$m^\star = \arg\min_m \max_x Var[x'] \tag{3.20}$$

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

$$\max_x Var[x'] = \frac{d}{m}\cdot\left(\frac{\beta-\alpha}{2}\cdot\frac{e^{\epsilon/m}+1}{e^{\epsilon/m}-1}\right)^2 \tag{3.21}$$

$$= C\cdot z\cdot\left(\frac{e^z+1}{e^z-1}\right)^2 = C\cdot z\cdot\coth^2(\frac{z}{2}) \tag{3.22}$$

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,

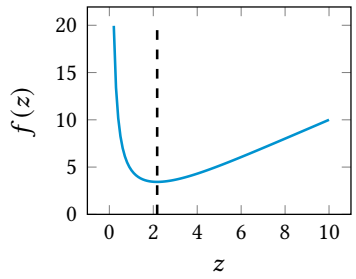


Figure 3.3: Plotting $f(z) = z \cdot \coth^2(\frac{z}{2})$. The gray dashed line indicate the location of the minimum.

minimizing (3.21) with respect to m is equivalent to minimizing (3.22) with respect to z, and then recover $m^\star$ as $\frac{\epsilon}{z^\star}$, where $z^\star$ is the optimal z minimizing (3.22). More formally:

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

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 3.3. Taking the derivative of $f(.)$ with respect to z and set it to zero gives us the minimum:

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

and then we have:

$$z = \frac{\coth(\frac{z}{2})}{\operatorname{csch}^2(\frac{z}{2})} = \frac{\sinh(z)}{2} \tag{3.23}$$

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:

$$m^\star = \max(1, \min(d, \left\lfloor \frac{\epsilon}{2.18} \right\rfloor)) \tag{3.24}$$

□

Proposition 3 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.

3.4.2 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):

$$\widehat{\mathbf{h}}_{\mathfrak{N}_v} = \mathtt{agg}\left(\{\mathbf{x}'_u, \forall u \in \mathfrak{N}_v\}\right) \tag{3.25}$$

$$\mathbf{h}_v = \mathtt{upd}\left(\widehat{\mathbf{h}}_{\mathfrak{N}_v}\right) \tag{3.26}$$

where $\widehat{\mathbf{h}}_{\mathfrak{N}_v}$ is the estimation of the first layer agg 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 agg 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 1. *Given a linear aggregator function, the aggregation defined by* (3.25) *is an unbiased estimation for* (3.1) *at layer* $l = 1$.

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

$$\mathbb{E}\left[\widehat{\mathbf{h}}_{\mathfrak{N}_v}\right] = \mathbf{h}_{\mathfrak{N}_v}$$

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

$$\mathbb{E}\left[\widehat{\mathbf{h}}_{\mathfrak{N}_v}\right] = \mathbb{E}\left[\mathtt{agg}\left(\{\mathbf{x}'_u, \forall u \in \mathfrak{N}_v\}\right)\right]$$

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

$$\mathbb{E}\left[\widehat{\mathbf{h}}_{\mathfrak{N}_v}\right] = \mathtt{agg}\left(\{\mathbb{E}\left[\mathbf{x}'_u\right], \forall u \in \mathfrak{N}_v\}\right)$$

Finally, by Proposition 1, we have:

$$\mathbb{E}\left[\widehat{\mathbf{h}}_{\mathfrak{N}_v}\right] = \mathtt{agg}\left(\{\mathbf{x}_u, \forall u \in \mathfrak{N}_v\}\right) = \mathbf{h}_{\mathfrak{N}_v}$$

□

The following proposition also shows the relationship of the estimation error in calculating the agg function and the neighborhood size $|\mathfrak{N}_v|$ for the special case of using the mean aggregator function:

Proposition 4. *Given the mean aggregator function for the first layer and* $\delta > 0$, *with probability at least* $1 - \delta$, *for any node* v, *we have:*

$$\max_{i \in \{1,\ldots,d\}} \left|(\widehat{\mathbf{h}}_{\mathfrak{N}_v})_i - (\mathbf{h}_{\mathfrak{N}_v})_i\right| = \mathcal{O}\left(\frac{\sqrt{d\log(d/\delta)}}{\epsilon\sqrt{|\mathfrak{N}_v|}}\right) \tag{3.27}$$

Proof. According to (3.10) and depending on Algorithm 2's output, for any node $u \in \mathcal{V}$ and any dimension $i \in \{1,2,\dots,d\}$, 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

$$C = \frac{d(\beta-\alpha)}{2m} \cdot \frac{e^{\epsilon/m}+1}{e^{\epsilon/m}-1} \tag{3.28}$$

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

$$\left| x'_{u,i} - x_{u,i} \right| \leq \frac{\beta-\alpha}{2} + C \tag{3.29}$$

and also by Proposition 1, we know that

$$\mathbb{E}\left[x'_{u,i} - x_{u,i} \right] = 0 \tag{3.30}$$

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

$$\begin{aligned} (\mathbf{h}_{\mathfrak{N}_v})_i &= \frac{1}{|\mathfrak{N}_v|} \sum_{u \in \mathfrak{N}_v} x_{u,i} \\ (\widehat{\mathbf{h}}_{\mathfrak{N}_v})_i &= \frac{1}{|\mathfrak{N}_v|} \sum_{u \in \mathfrak{N}_v} x'_{u,i} \end{aligned} \tag{3.31}$$

Considering (3.29) to (3.31) and using the Bernstein inequality, we have:

$$\begin{aligned} &\Pr\left[\left| (\widehat{\mathbf{h}}_{\mathfrak{N}_v})_i - (\mathbf{h}_{\mathfrak{N}_v})_i \right| \geq \lambda \right] \\ &= \Pr\left[\left| \sum_{u \in \mathfrak{N}_v} (x'_{u,i} - x_{u,i}) \right| \geq \lambda |N(v)| \right] \\ &\leq 2 \cdot \exp\left\{ - \frac{\lambda^2 |\mathfrak{N}_v|}{\frac{2}{|\mathfrak{N}_v|} \sum_{u \in \mathfrak{N}_v} Var[x'_{u,i} - x_{u,i}] + \frac{2}{3}\lambda(\frac{\beta-\alpha}{2} + C)} \right\} \\ &= 2 \cdot \exp\left\{ - \frac{\lambda^2 |\mathfrak{N}_v|}{2 Var[x'_{u,i}] + \frac{2}{3}\lambda(\frac{\beta-\alpha}{2} + C)} \right\} \end{aligned} \tag{3.32}$$

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

$$Var[x'_{u,i}] = \frac{m}{d}C^2 - \left(x_{v,i} - \frac{\alpha+\beta}{2}\right)^2 \tag{3.33}$$

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

$$C = \frac{d(\beta-\alpha)}{2m}\mathcal{O}(\frac{m}{\epsilon}) = \mathcal{O}(\frac{d}{\epsilon}) \tag{3.34}$$

and therefore we have:

$$Var[x'_{u,i}] = \frac{m}{d}\left(\mathcal{O}(\frac{d}{\epsilon})\right)^2 - \left(x_{v,i} - \frac{\alpha+\beta}{2}\right)^2 = \mathcal{O}(\frac{md}{\epsilon^2}) \tag{3.35}$$

Substituting (3.34) and (3.35) in (3.32), we have:

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

According to the union bound, we have:

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

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

$$\delta = 2d \cdot \exp\left\{-\frac{\lambda^2|\mathfrak{N}_v|}{\mathcal{O}(\frac{md}{\epsilon^2}) + \lambda\mathcal{O}(\frac{d}{\epsilon})}\right\} \tag{3.36}$$

Solving the above for λ, we get:

$$\lambda = \mathcal{O}\left(\frac{\sqrt{d\log(d/\delta)}}{\epsilon\sqrt{|\mathfrak{N}_v|}}\right) \tag{3.37}$$

□

Proposition 4 indicates that with the mean aggregator function (which can be extended to other agg 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 $\mathfrak{N}_v$ is usually small in real graphs, which hinders the agg function from driving out the injected noise on its own.

Algorithm 3: KProp Layer

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

Output : Embedding vector $\mathbf{h}_v, \forall v \in \mathcal{V}$

for all $v \in \mathcal{V}$ **do in parallel**
  $\mathbf{h}^0_{\mathfrak{N}_v} = \mathbf{x}_v$
  for $k = 1$ *to* K **do**
    $\mathbf{h}^k_{\mathfrak{N}_v} = \mathtt{agg}\left(\{\mathbf{h}^{k-1}_{\mathfrak{N}_u}, \forall u \in \mathfrak{N}_v - \{v\}\}\right)$
  end
  $\mathbf{h}_v = \mathbf{h}^K_{\mathfrak{N}_v}$
end
return $\{\mathbf{h}_v, \forall v \in \mathcal{V}\}$

In a different context, prior works have shown that considering higher-order neighbors can help learn better node representations (Abu-El-Haija et al., 2019; Klicpera et al., 2019; Morris et al., 2019). 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 3, 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 agg 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 3.2, 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 agg function for the backbone model, as KProp already uses a linear agg that satisfies Corollary 1; 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 (Q. Li et al., 2018). 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 $\mathfrak{N}_v - \{v\}$, i.e., we do not include self-loops. While it has been shown that adding self-loops can improve accuracy in conventional GNNs (Kipf & Welling, 2017), 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.

3.4.3 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 $\mathcal{V}_{\mathcal{L}}$:

$$\ell\left(\mathbf{y}, \hat{p}(\mathbf{y} \mid \mathbf{x})\right) = -\sum_{v \in \mathcal{V}_{\mathcal{L}}} \mathbf{y}_v^T \log \hat{p}(\mathbf{y} \mid \mathbf{x}_{\mathbf{v}}) \tag{3.38}$$

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 (McPherson et al., 2001), meaning that nodes with structural similarity tend to have similar labels (H. Wang & Leskovec, 2020). 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 (Kairouz et al., 2016), Unary Encoding (T. Wang et al., 2017), or Local Hashing (T. Wang et al., 2017). In this chapter, 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 (T. Wang et al., 2017); 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:

$$p(\mathbf{y}' \mid \mathbf{y}) = \begin{cases} \frac{e^{\epsilon}}{e^{\epsilon}+c-1}, & \text{if } \mathbf{y}' = \mathbf{y} \\ \frac{1}{e^{\epsilon}+c-1}, & \text{otherwise} \end{cases} \tag{3.39}$$

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 chapter, however, we instead use the

GCN aggregator function (Kipf & Welling, 2017):

$$\mathtt{agg}\left(\{\mathbf{y}'_u, \forall u \in \mathfrak{N}_v\}\right) = \sum_{u \in \mathfrak{N}_v} \frac{\mathbf{y}'_u}{\sqrt{|\mathfrak{N}_u| \cdot |\mathfrak{N}_v|}} \tag{3.40}$$

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 $\sqrt{|\mathfrak{N}_u| \cdot |\mathfrak{N}_v|}$, while for the mean, it is $|\mathfrak{N}_v| = \sqrt{|\mathfrak{N}_v| \cdot |\mathfrak{N}_v|}$. 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 $\hat{p}(\mathbf{y} \mid \mathbf{x})$:

$$\hat{p}(\mathbf{y}' \mid \mathbf{x}) = \sum_{\mathbf{y}} p(\mathbf{y}' \mid \mathbf{y}) \cdot \hat{p}(\mathbf{y} \mid \mathbf{x}) \tag{3.41}$$

where $p(\mathbf{y}' \mid \mathbf{y})$ is directly obtained from (3.39). 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 $\hat{p}(\tilde{\mathbf{y}} \mid \mathbf{x})$:

$$\hat{p}(\tilde{\mathbf{y}} \mid \mathbf{x}) = softmax\big(h\big(\hat{p}(\mathbf{y}' \mid \mathbf{x}), K_y\big)\big) \tag{3.42}$$

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 $\ell\left(\mathbf{y}', \hat{p}(\mathbf{y}' \mid \mathbf{x})\right)$, as Patrini et al. (2017) show that this loss function, which they call forward correction loss, is unbiased, meaning that under expected label noise, $\ell\left(\mathbf{y}', \hat{p}(\mathbf{y}' \mid \mathbf{x})\right)$ is equal to $\ell\left(\mathbf{y}, \hat{p}(\mathbf{y} \mid \mathbf{x})\right)$, 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 4, 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 2. *Algorithm 4 satisfies $(\epsilon_x + \epsilon_y)$-local differential privacy for graph nodes.*

Proof. The training steps in Theorem 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 4 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 (Dwork, Roth, et al., 2014), Algorithm 4 satisfies $(\epsilon_x + \epsilon_y)$-LDP for each node. □

Corollary 2 shows that the entire training procedure is LDP due to the robustness of differential privacy to post-processing (Dwork, Roth, et al., 2014). Furthermore, any prediction performed by the LPGNN is again subject to the post-processing theorem (Dwork, Roth, et al., 2014), and therefore, satisfies LDP for the nodes, as the LDP mechanism is applied to the private data only once.

3.5 Experimental Setup

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.

3.5.1 Datasets

We used two different sets of publicly available real-world datasets: two citation networks, Cora and Pubmed (Yang et al., 2016), which have a lower average degree, and two social networks, Facebook (Rozemberczki et al., 2019), and LastFM (Rozemberczki & Sarkar, 2020) that have a higher average degree. The description of the datasets is as followed:

- *Cora and Pubmed (Yang et al., 2016):* 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 (Rozemberczki et al., 2019):* 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.

- *LastFM (Rozemberczki & Sarkar, 2020):* 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.

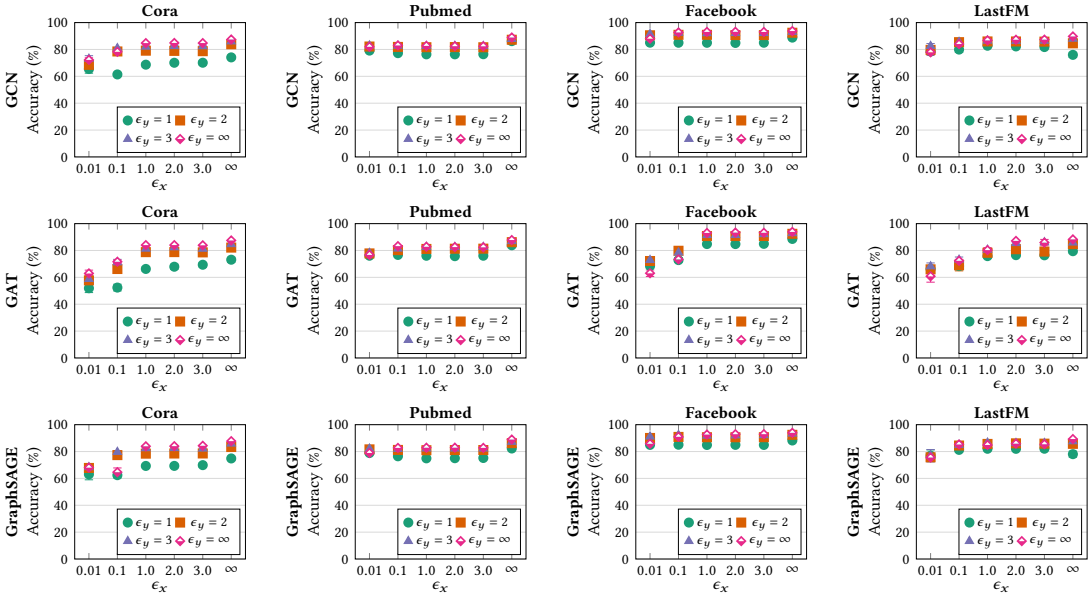


Figure 3.4: Comparison of LPGNN's performance with different GNN model under varying feature and label privacy budgets.

Summary statistics of the datasets are provided in Table 3.1.

3.5.2 Experiment Settings

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 (Kipf & Welling, 2017), GAT (Veličković et al., 2018), and GraphSAGE (Hamilton et al., 2017a), 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 (Klambauer et al., 2017) 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 3.4 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

[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.

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 (Kingma & Ba, 2014) 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.

3.5.3 Hardware and Software

The experiments were run on a remote server with Intel Xeon 6238 CPU and 60GB of available RAM, which was part of a Sun Grid Engine infrastructure. All the models were trained using a single NVIDIA V100 GPU having 32GB of graphic memory. The code were written in PyTorch using the PyTorch Geometric library (Fey & Lenssen, 2019).

3.6 Results

3.6.1 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 3.4.

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 (as in (3.40)) 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.

3.6.2 Analyzing the Multi-Bit Mechanism

In Table 3.2, 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) (Balle & Wang, 2018). The 1-bit mechanism (Ding et al., 2017), is obtained by setting $m = d$ in Algorithm 2. 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 LDP, namely (ϵ, δ)-LDP, which (loosely speaking) means that it satisfies ϵ-LDP with probability at least $1 - \delta$. Here, we use the Analytic Gaussian mechanism (Balle & Wang, 2018), 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 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 3.3, 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 (K. Xu et al., 2019), 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.

3.6.3 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 3.5. 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 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

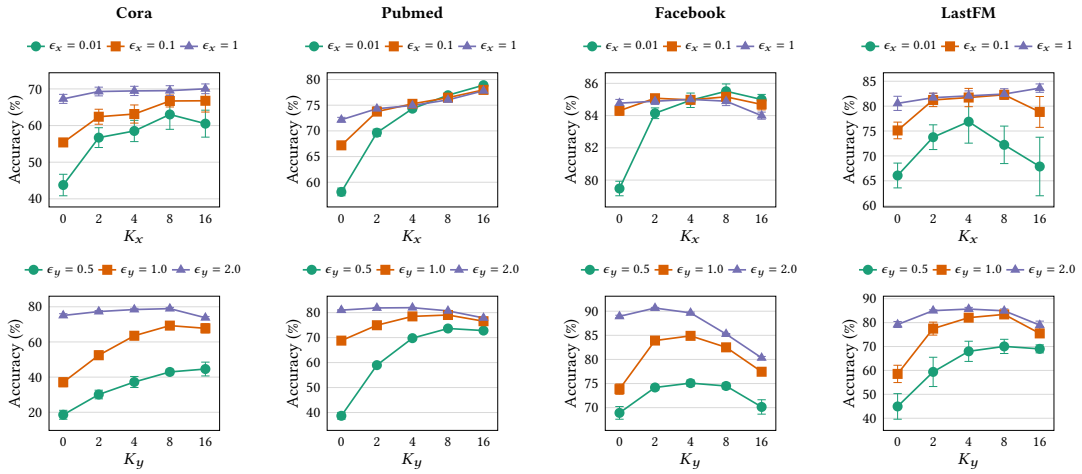


Figure 3.5: Effect of the KProp step parameter on the performance of LPGNN. The top row depicts the effect of feature KProp with $\epsilon_y = 1$. The bottom row shows the effect of label KProp with $\epsilon_x = 1$. The y-axis is not set to zero to focus on the trends.

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.

3.6.4 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 (Patrini et al., 2017), described in Section 3.4. 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 3.4 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.

3.7 Discussion

3.7.1 Model Robustness

Not all kinds of deep learning models are robust to differentially private input perturbations, as the injected noise will usually go through multiple layers of non-linearity, degrading the utility of the models significantly (T. Zhu et al., 2020). However, we systematically showed that this is not the case for GNNs. In these models, the noise added to the data will average out in the neighborhood aggregation before applying the non-linear transformation, which helps them preserve their robustness against differentially private input perturbations. As a result, GNN-based deep learning models can be used not only to solve the introduced node data privacy problem, but also as an alternative solution to other privacy-preserving approaches to address more general supervised learning tasks in which the training data cannot leave users' devices due to privacy constraints.

3.7.2 Other Solutions

The problem setting of node data privacy is very similar to federated learning, and despite the existing connectivity between the nodes in our problem, with some considerations, federated learning can also be used here as a solution. To this end, the server has to send the GNN model to every node (user) of the graph to train the GNN locally using their private features and then send the model updates back to the server for aggregation. But for locally training the GNN, each user requires the feature vectors of their neighbors to evaluate the first layer's $\mathtt{agg}$ function. Assuming an end-to-end communication is possible between users (e.g., via a secure channel established through the server), this evaluation should be done using a secure multi-party computation (SMC) protocol, such as secret sharing (Shamir, 1979), to protect the privacy of users. Although the result of the first layer aggregation can be saved by each user to be used in subsequent training iterations (as they do not depend on the GNN parameters), still to compute the $\mathtt{agg}$ functions in other layers, every two adjacent users have to exchange their representation vectors, and this has to be repeated at every training iteration, which incurs a huge communication cost. More specifically, in addition to the usual $\mathcal{O}(|\mathcal{V}|)$ communication required for model parameter exchange between users and the server, $\mathcal{O}(|\mathcal{E}|)$ communication is also needed at each round for vector exchange between neighbors. Consequently, federated learning is not feasible in practice due to its excessive communication overhead.

Split learning (Gupta & Raskar, 2018) is another related technique that can be employed to

address node data privacy. In split learning, the users do not train the entire model locally, but the neural network is split between the users and the server, where the users initiate the forward pass up to a specific layer, called the cut layer, whose output is sent to the server to complete the forward pass on the rest of the layers. The gradients are similarly back-propagated from the server at the cut layer to the users to complete the process. In our problem, at least the first layer of the GNN, including both the agg and the upd functions, have to be run at the user-side to ensure that the server cannot easily recover the private features, which again have the challenges of evaluating the agg function at the first layer, similar to the federated learning. While, as mentioned in the case of federated learning, this is a one-time operation, evaluating a function using SMC usually requires multiple rounds of data exchange among the users so that the overall communication could be cumbersome for web-scale graphs.

Nevertheless, neither split learning nor federate learning provides any provable privacy guarantee on their own. Unless combined with other privacy-enhancing technologies, such as differential privacy, they are vulnerable to potential privacy attacks, such as membership inference or model inversion (B. Liu et al., 2020; Mirshghallah et al., 2020). Conversely, our locally private approach not only is very efficient in terms of communication cost but also comes with strong privacy guarantees and at the same time maintains a sustainable level of accuracy.

3.7.3 Implications

The research presented in this chapter contributes toward developing privacy-preserving GNNs, which is important as most of today's real-world graphs, such as social networks, are generated from people's activities or represent human-related interactions. Using the method proposed in this chapter, we encourage a wide range of applications, from social network and dating apps to shopping and financial applications that use GNNs as part of their systems, to significantly reduce their dependency on collecting users' personal raw data, and instead switch to the differentially private data collection mechanisms. This not only provides users with provable data privacy guarantees, but also has an impact on the security of the application servers, as there would no longer be a huge volume of users' private information on the server's data centers, leaving potential attackers no or less incentive to breach their systems. From the user's perspective, the proposed method will allow them to have more control on their private data, for example by deciding whether to contribute with their data based on the privacy budget parameter ϵ declared by the server, rather than just giving required permissions to the application to get access to their private data. But indeed, moving from current data-hungry methods to privacy-preserving technologies will certainly result in the degradation of the system's utility, in terms of accuracy or other performance measures, which might adversely affect the user experience. However, as we witnessed, our method could maintain an acceptable level of privacy-utility trade-off, so that the overall performance of the system does not decline considerably, and at the same time, it provides an

appropriate level of data privacy protection to the users.

3.8 Conclusion

In this chapter, 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 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 chapter, 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. Thirdly, we used the same number of features to be perturbed by the multi-bit mechanism for all the nodes, whose value was obtained by minimizing the worst-case variance of the multi-bit estimator. As future work, we investigate an optimized heterogeneous feature perturbation scheme by minimizing a variance-based loss function over the entire network. 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.

Algorithm 4: 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 2.
if *current node is in* $\mathcal{V}_{\mathcal{L}}$ **then**
  Obtain a perturbed label $\mathbf{y}'$ by (3.39).
else
  $\mathbf{y}' \leftarrow \vec{\mathbf{0}}$
end
Send $(\mathbf{x}^*, \mathbf{y}')$ to the server.

Server-side:
Obtain $\mathbf{x}'_v$ using (3.10) 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 (3.41)
    Obtain $\hat{p}(\tilde{\mathbf{y}} \mid \mathbf{x}_v)$ using (3.42)
  end
  $\mathbf{W}^{t+1} \leftarrow \mathbf{W}^t - \eta \nabla \sum_{v \in {\mathcal{V}_{\mathcal{L}}}^{tr}} \ell\left(\tilde{\mathbf{y}}_v, \hat{p}(\tilde{\mathbf{y}} \mid \mathbf{x}_v)\right)$
  $\ell^t_{val} \leftarrow \sum_{v \in {\mathcal{V}_{\mathcal{L}}}^{val}} \ell\left(\mathbf{y}'_v, \hat{p}(\mathbf{y}' \mid \mathbf{x}_v)\right)$
  $Acc^t_{val} \leftarrow \frac{1}{|{\mathcal{V}_{\mathcal{L}}}^{val}|} \sum_{v \in {\mathcal{V}_{\mathcal{L}}}^{val}} Accuracy(\hat{p}(\mathbf{y} \mid \mathbf{x}_v), \mathbf{y}'_v)$
  $Acc^t_{tr} \leftarrow \frac{1}{|{\mathcal{V}_{\mathcal{L}}}^{tr}|} \sum_{v \in {\mathcal{V}_{\mathcal{L}}}^{tr}} Accuracy(\hat{p}(\mathbf{y} \mid \mathbf{x}_v), \mathbf{y}'_v)$
end
$t \leftarrow \operatorname{argmin}_t \ell^t_{val}$ such that $Acc^t_{tr} \leq Acc^*$ and $Acc^t_{val} \leq Acc^*$
return $\mathbf{W}^t$

Table 3.1: Descriptive Statistics of the Used Datasets

Dataset	No. Classes	No. Nodes	No. Edges	No. Features	Avg. Degree
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

Table 3.2: Accuracy of LPGNN with different LDP mechanisms ($\epsilon_y = \infty$)

Dataset	Mechanism	$\epsilon_x = 0.01$	$\epsilon_x = 0.1$	$\epsilon_x = 1$	$\epsilon_x = 2$
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**

Table 3.3: 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

Table 3.4: 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**

4 Private Graph Neural Networks with Aggregation Perturbation

In Chapter 3, we studied the problem of learning Graph Neural Networks (GNNs) with Local Differential Privacy, where only the node-specific data, such as features and labels, were considered to be locally private, while the relationship links, or edges, remained publicly accessible to a central server responsible for GNN training.

In this chapter, we are shifting our focus towards exploring privacy-preserving GNNs from a different perspective: global (or central) model of Differential Privacy (DP). In this context, every piece of graph data, including node features, labels, and also the interconnected edges, is treated as private information that needs to be protected. Our aim here is to release a GNN model, trained with differential privacy guarantees.

We propose a novel differentially private **G**NN based on **A**ggregation **P**erturbation (GAP), which adds stochastic noise to the GNN's aggregation function to statistically obfuscate the presence of a single edge (edge-level privacy) or a single node and all its adjacent edges (node-level privacy). Tailored to the specifics of private learning, GAP's new architecture is composed of three separate modules: (i) the encoder module, where we learn private node embeddings without relying on the edge information; (ii) the aggregation module, where we compute noisy aggregated node embeddings based on the graph structure; and (iii) the classification module, where we train a neural network on the private aggregations for node classification without further querying the graph edges.

GAP's major advantage over previous approaches is that it can benefit from multi-hop neighborhood aggregations, and guarantees both edge-level and node-level DP not only for training, but also at inference with no additional costs beyond the training's privacy budget. We analyze GAP's formal privacy guarantees and conduct empirical experiments over three real-world graph datasets. We demonstrate that GAP offers significantly better accuracy-privacy trade-offs than state-of-the-art DP-GNN approaches and naive MLP-based baselines.

This chapter will delve into the details of GAP, its formal privacy guarantees, and empirical experiments over three real-world graph datasets. Section 4.1 offers an in-depth explanation

of the challenges and our contributions in learning differentially private GNNs. We discuss the related work in Section 4.2, and then Section 4.3 gives a formal definition of the problem and the necessary background information. Next, our proposed differentially private GNN model is detailed in Section 4.4. The privacy aspects of our approach are formally analyzed in Section 4.5, while Section 4.6 delves into various facets and limitations of our proposed method. The experimental setup and results are elaborated in Section 4.7 and Section 4.8, respectively. Finally, Section 4.9 concludes the chapter.

The work presented in this chapter was originally published in:

- Sajadmanesh, S., Shamsabadi, A. S., Bellet, A., & Gatica-Perez, D. (2023). Gap: Differentially private graph neural networks with aggregation perturbation. *32nd USENIX Security Symposium (USENIX Security 23)*

4.1 Introduction

Real-world datasets are often represented by graphs, such as social (Qiu et al., 2018), financial (J. Wang et al., 2021), transportation (Diao et al., 2019), or biological (Kearnes et al., 2016) networks, modeling the relations (i.e., edges) between a collection of entities (i.e., nodes). Graph Neural Networks (GNNs) have achieved state-of-the-art performance in learning over such relational data in various graph-based machine learning tasks, such as node classification, link prediction, and graph classification (Corso et al., 2020; Hamilton et al., 2017a; Kipf & Welling, 2017; K. Xu et al., 2019; M. Zhang & Chen, 2018; M. Zhang et al., 2021). Due to their superior performance, GNNs are now widely used in many applications, such as recommendation systems, credit issuing, traffic forecasting, drug discovery, and medical diagnosis (Cheung & Moura, 2020; Gkalelis et al., 2021; W. Jiang & Luo, 2022; Mohammadshahi & Henderson, 2020; Ying et al., 2018).

Privacy concerns. Despite their success, real-world deployments of GNNs raise privacy concerns when graphs contain personal data: for instance, social or financial networks involve sensitive information about individuals and their interactions. Recent works (X. He, Jia, et al., 2021a; X. He, Wen, et al., 2021; Olatunji, Nejdl, & Khosla, 2021; F. Wu et al., 2021) have extended the study of the privacy leakage of standard deep learning models to GNNs, showing the risk of information leakage regarding training data is even higher in GNNs, as they incorporate not only node features and labels but also the graph structure itself (Duddu et al., 2020). Consequently, GNNs are vulnerable to various privacy attacks, such as node membership inference (X. He, Wen, et al., 2021; Olatunji, Nejdl, & Khosla, 2021) and edge stealing (X. He, Jia, et al., 2021a; F. Wu et al., 2021). For example, a GNN trained on a social network for friendship recommendation could reveal the existing relationships between the users via its predictions. As another example, a GNN trained on the social graph of COVID-19 patients can be used by government authorities to predict the spread of the disease, but an adversary may recover

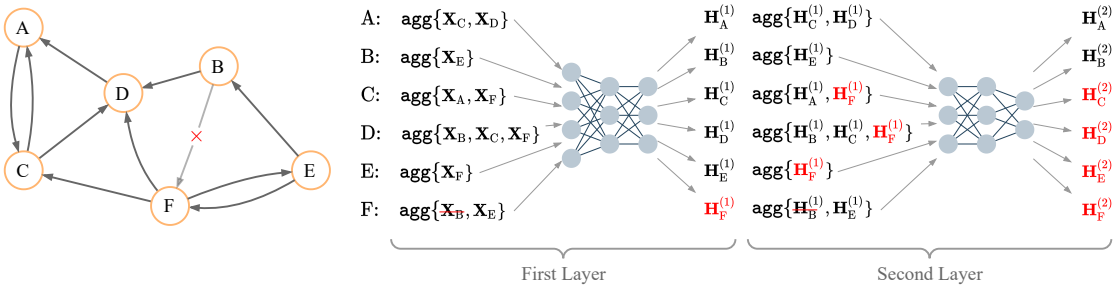
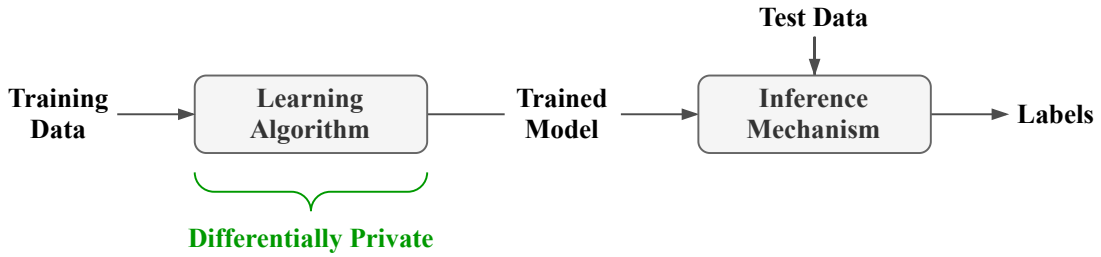


Figure 4.1: Schema of an unfolded 2-layer GNN taking an example graph as input. At each layer, every node aggregates its neighbors' embedding vectors (initially node features, e.g. $\mathbf{X}_A$ for node A), which is then updated using a neural net into a new vector (e.g., $\mathbf{H}_A$). Removing an arbitrary edge (here, the edge from node B to F) excludes the source node (B) from the aggregation set of the destination node (F). At the first layer, this will only alter the destination node's embedding, but this change is propagated to the neighboring nodes in the next layer. Node embeddings that are affected by the removal of edge (B,F) are indicated in red.

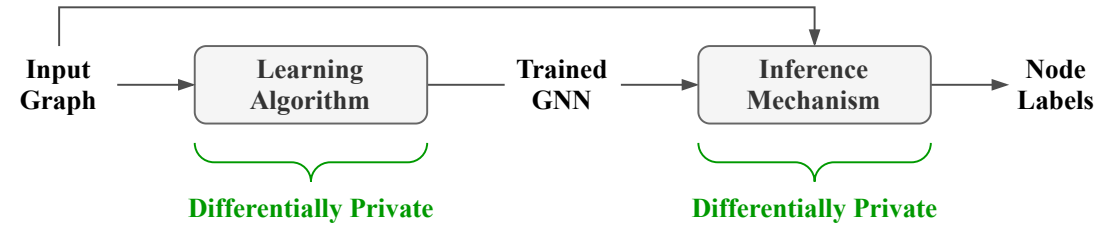
private information about the participating patients.

Problem and motivation. Motivated by these privacy concerns, we investigate the problem of designing privacy-preserving GNNs for private, sensitive graphs. Our goal is to protect the sensitive graph structure and other accompanying data using the framework of Differential Privacy (DP) (Dwork, 2008).In the context of graphs, two different variants of DP have been defined: *edge-level* and *node-level* DP (Raskhodnikova & Smith, 2016). Informally, an edge-level ϵ-DP algorithm have roughly the same output (as measured by ϵ) if one edge is removed from the input graph. This ensures that the algorithm's output does not reveal the existence of a particular edge in the graph. Correspondingly, node-level private algorithms conceal the presence of a particular node together with all its associated edges and attributes. Clearly, node-level DP is a stronger privacy definition, but it is harder to attain because it requires the algorithm's output distribution to hide much larger differences in the input graph.

Challenges. As GNNs utilize the structural information in the graph data, protecting data privacy in such models is more challenging than in standard ones. As shown in Figure 4.1, one of these challenges is the interdependency between the node embeddings resulting from the GNN's data aggregation mechanism. Specifically, a K-layer GNN iteratively learns node embeddings by aggregating information from every node's K-hop neighborhood (i.e., from nodes that are at a distance at most K in the graph). Hence, the embedding of a node is influenced not only by the node itself but also by all the nodes in its K-hop proximity. This fact voids the privacy guarantees of standard DP learning paradigms, such as DP-SGD (Abadi et al., 2016), as the training loss of GNNs can no longer be decomposed into individual samples. Furthermore, the number of interdependent embeddings grows exponentially with K, hindering the ability of a DP solution to hide the output differences effectively. Therefore,



(a) Learning standard DNNs with DP



(b) Learning GNNs with DP

Figure 4.2: Comparison of DP learning with (a) conventional deep neural networks, and (b) graph neural networks. Given the trained model, the inference mechanism of a DNN is independent of the training data, so a DP learning algorithm implies a DP inference mechanism as well. With GNNs however, graph data is queried again at inference time, so the inference step requires specific attention to be made differentially private.

how to get more representational power from higher-order GNN aggregations while ensuring DP is an important challenge to address.

Another major challenge is to guarantee *inference privacy*, i.e., preserving the privacy of graph data not only for training but also at inference time, when the trained GNN model is queried to make predictions for test nodes. Unlike conventional deep learning models, where the training data is not reused at inference time, the inference about any node in a K-layer GNN requires aggregating data from its K-hop neighborhood, which can reveal information about the neighboring nodes. Therefore, private graph data can still be leaked at inference time, even with privately trained model parameters. As a result, it is critical to ensure that both the training and inference stages of a GNN satisfy DP. This is illustrated in Figure 4.2.

Our contributions. To address the above challenges, we propose GAP, a privacy-preserving GNN model satisfying edge-level privacy, which is also extensible to node-level privacy if combined with standard private learning algorithms such as DP-SGD. As perturbing an edge in the input graph can practically be viewed as changing a sample in a node's neighborhood aggregation set, GAP preserves edge privacy via *aggregation perturbation*: we add calibrated Gaussian noise to the output of the aggregation function, which can effectively hide the presence of a single edge (edge-level privacy) or a group of edges (node-level privacy). To avoid

accumulating privacy costs at every model update, we propose a custom GNN architecture (Figure 4.3) comprising three individual components: (i) the encoder module, where we pre-train an encoder to extract lower-dimensional node features without relying on the graph structure; (ii) the aggregation module, where we use aggregation perturbation to privately compute multi-hop aggregated node embeddings using the graph edges and the encoded features; and (iii) the classification module, where we train a neural network on the aggregated data for node classification without further querying the graph edges.

Aggregation perturbation allows us to benefit from higher-order, multi-hop aggregations by composing individual noisy aggregations, yet the proposed architecture significantly reduces the privacy costs as the perturbed aggregations are computed once on lower-dimensional embeddings, and reused during training and inference. GAP also provides inference privacy, as the inference of any node relies on the perturbed aggregations, which hide information about neighboring nodes. Due to reusing cached aggregations, the inference step does not incur additional privacy costs beyond that of training.

Results. We analyze GAP's formal privacy guarantees using Rényi Differential Privacy (Mironov, 2017), and empirically evaluate its accuracy-privacy performance on three medium to large-scale graph datasets, namely Facebook, Reddit, and Amazon. We demonstrate that GAP's accuracy surpasses the competing baselines' at (very) low privacy budgets under both edge-level DP (e.g., $\epsilon \geq 0.1$ on Reddit) and node-level DP (e.g., $\epsilon \geq 1$ on Reddit), and observe that it always performs on par or better than a naive (privately trained) MLP model which does not utilize the graph's structural information.

4.2 Related Work

Graph neural networks. Deep learning on graphs has emerged in the past few years to tackle different kinds of graph-based learning tasks. A variety of GNN models and various architectures have been proposed, including Graph Convolutional Networks (Kipf & Welling, 2017), Graph Attention Networks (Veličković et al., 2018), GraphSAGE (Hamilton et al., 2017a), Graph Isomorphism Networks (K. Xu et al., 2019), Jumping Knowledge Networks (K. Xu et al., 2018), and Gated Graph Neural Networks (Y. Li et al., 2016). For the latest advances and trends in GNNs, we refer the reader to the available surveys (Abadal et al., 2021; Hamilton et al., 2017b; Z. Wu et al., 2020; Z. Zhang, Cui, & Zhu, 2022; Zhou, Cui, et al., 2020).

Privacy attacks on GNNs. Several recent works have investigated the possibility of performing privacy attacks against GNNs and quantified the privacy leakage of publicly released GNN models or node embeddings trained on private graph datasets. These attacks mainly fall in three categories: attribute inference, membership inference, and link stealing/graph reconstruction. Duddu et al. (2020) conduct a comprehensive study on quantifying the privacy leakage of graph embedding algorithms trained on sensitive graph data and propose various

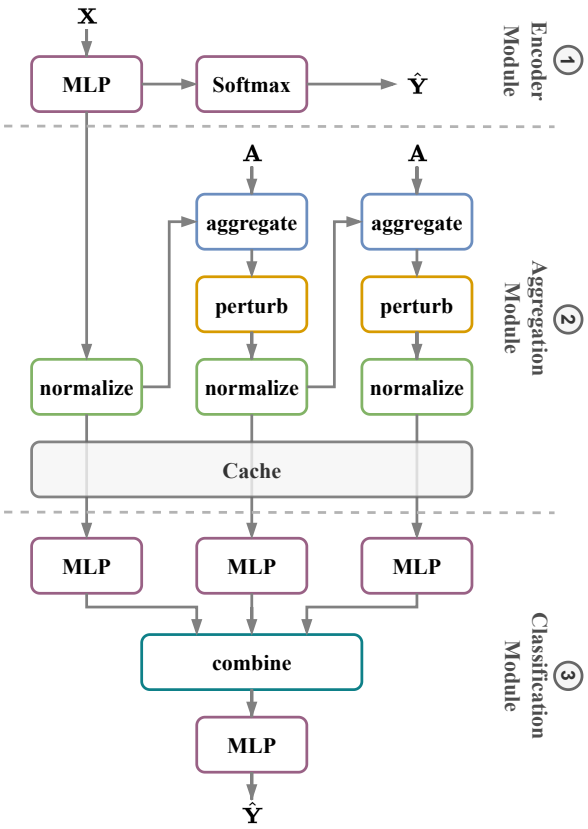


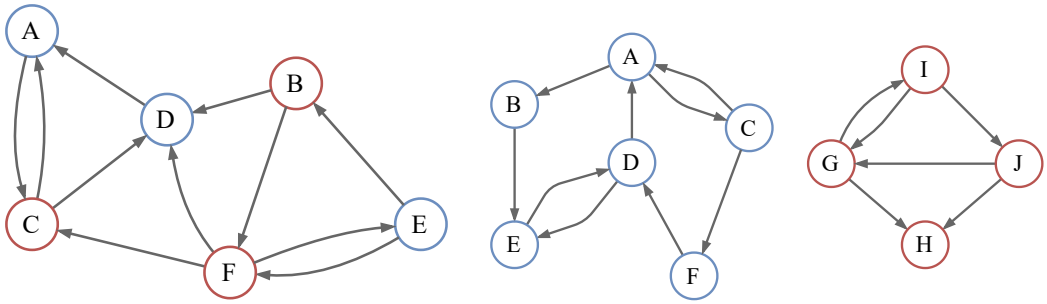
Figure 4.3: Overview of GAP's architecture: (1) The encoder is trained using only node features (**X**) and labels (**Y**). (2) The encoded features are given to the aggregation module to compute private K-hop aggregations (here, $K = 2$) using the graph's adjacency matrix (**A**). (3) The classification module is trained over the private aggregations for label prediction.

privacy attacks on GNNs under practical threat models and adversary assumptions. Z. Zhang, Chen, et al. (2022) study the information leakage in graph embeddings and propose three different inference attacks against GNNs: inferring graph properties (such as number of nodes and edges), inferring whether a given subgraph is contained in the target graph, and graph reconstruction with similar statistics to the target graph. X. He, Jia, et al. (2021a) propose a series of black-box link stealing attacks on GNN models, and show that an adversary can accurately infer a link between any pair of nodes in a graph used to train the GNN. Z. Zhang et al. (2021b) study the connection between model inversion risk and edge influence, and show that edges with greater influence are more likely to be inferred. F. Wu et al. (2021) also study the link stealing attack via influence analysis, and propose an effective attack against GNNs based on the node influence information. The feasibility of the membership inference attack against GNNs has also been studied and several attacks with different threat models have been proposed in the literature (Bang et al., 2021; Duddu et al., 2020; X. He, Wen, et al., 2021; Olatunji, Nejdl, & Khosla, 2021). Overall, these works underline the privacy risks of GNNs trained on sensitive graph data and confirm the vulnerability of these models to various

privacy attacks.

Privacy-preserving GNNs. There have been attempts to mitigate the above-mentioned attacks by proposing privacy-preserving GNN models. K. Li et al. (2020) presents a graph adversarial training framework that integrates disentangling and purging mechanisms to remove users' private attributes from learned node representations. Liao et al. (2021) also follow an adversarial learning approach to address the attribute inference attack on GNNs, where they introduce a mini-max game between the desired graph feature encoder and the worst-case attacker. B. Wang et al. (2021) propose a privacy-preserving GNN learning framework from the mutual information perspective, where they learn node representations, such that the primary learning task achieves high performance, while the privacy protection task performs arbitrarily bad. Hsieh and Li (2021) also propose an adversarial defense mechanism against attribute inference attacks on GNNs by maintaining the accuracy of target label classification and reducing the accuracy of private label classification. There is also increasing trend in using federated and split learning to address the privacy issues of GNNs in distributed learning settings (C. Chen et al., 2021; F. Chen et al., 2021; C. He, Balasubramanian, et al., 2021; Z. Liu et al., 2021; Meng et al., 2021; Ni et al., 2021; Pei et al., 2021; B. Wang et al., 2020; C. Wu et al., 2021; Xie et al., 2021; H. Zhang et al., 2021; K. Zhang et al., 2021). However, none of the aforementioned works employ the notion of DP, and thus they do not provide provable privacy guarantees.

Differentially private GNNs. Very recently, DP has also been used to provide formal privacy guarantees in various GNN learning settings. Sajadmanesh and Gatica-Perez (2021) propose a locally private GNN model by considering a distributed learning setting, where node features and labels are private but training the GNN is federated by a central server with access to graph edges. They use local DP to protect node data and design a learning algorithm utilizing the public graph structure for feature and label denoising. However, their method cannot be used in applications where the graph edges are private. F. Wu et al. (2021) propose an edge-level DP learning algorithm for GNNs by perturbing the input graph directly using either randomized response (called EDGERAND) or the Laplace mechanism (called LAPGRAPH). Then, a GNN is trained over the resulting noisy graph. However, their method cannot be extended trivially to the node-level privacy setting. Olatunji, Funke, and Khosla (2021) consider a centralized learning setting and propose a node-level private GNN by adapting the framework of PATE (Papernot et al., 2016). They train the student GNN model using public graph data, which is privately labeled using the teacher GNN models trained exclusively for each query node. However, their dependence on public graph data restricts the applicability of their method. Daigavane et al. (2021) also propose a node-level private approach for training 1-layer GNNs by extending the standard DP-SGD algorithm and privacy amplification by subsampling results to bounded-degree graph data. However, their approach fails to provide inference privacy and



(a) Transductive Learning (b) Inductive Learning

Figure 4.4: (a) Transductive learning: training and inference steps are conducted on the same graph, but different nodes are used for training and testing. Here, the blue nodes (A, D, and E) are used for training and the red nodes (B, C, and F) for inference. (b) Inductive learning: training and inference steps are performed on different graphs. Here, the left and right graphs are used for training and inference, respectively.

is limited to 1-layer GNNs and thus cannot leverage higher-order aggregations.[1]

Comparison with existing methods. To our best knowledge, GAP is the first approach providing both edge-level or node-level privacy guarantees based on the application requirements. Unlike existing methods, our approach does not rely on public data, can leverage multi-hop aggregations beyond first-order neighbors, and guarantees inference privacy at no additional cost. In Section 4.7, we also show that GAP outperforms other baselines in terms of accuracy-privacy trade-off.

4.3 Background and Problem Formulation

4.3.1 Graph Neural Networks

GNNs aim to learn a representation for every node in the input graph by incorporating the initial node features and the graph structure (edges). The learned node representations, or embeddings, can then be used for the downstream machine learning task. In this chapter, we focus on node classification, where the embeddings are used to predict the label of the graph nodes. Node-wise prediction problems can be tackled in either transductive or inductive setting. In the transductive setting, both training and testing are performed on the same graph, but different nodes are used for training and testing. Conversely, in the inductive setting, training and testing are performed on different graphs. This is illustrated in Figure 4.4.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X}, \mathbf{Y})$ be an unweighted directed graph dataset consisting of sets of nodes $\mathcal{V}$ and edges $\mathcal{E}$ represented by a binary adjacency matrix $\mathbf{A} \in \{0,1\}^{N \times N}$, where $N = |\mathcal{V}|$ denotes the

[1] After releasing our article, Daigavane et al. (2021) extended their method to support more than one message-passing layers.

number of nodes, and $\mathbf{A}_{i,j} = 1$ if there is a directed edge $(i, j) \in \mathscr{E}$ from node i to node j. Nodes are characterized by d-dimensional feature vectors stacked up in an $N \times d$ matrix $\mathbf{X}$, where $\mathbf{X}_v$ denotes the feature vector of the v-th node. $\mathbf{Y} \in \{0,1\}^{N \times C}$ represents the labels of the nodes, where $\mathbf{Y}_v$ is a C-dimensional one-hot vector denoting the label of the v-th node, and C is the number of classes. Note that in the transductive learning setting, only a subset $\mathscr{V}_T \subset \mathscr{V}$ of the nodes is labeled, and thus $\mathbf{Y}_v$ is a zero vector for all $v \notin \mathscr{V}_T$.

A typical K-layer GNN consists of K sequential graph convolution layers. Layer i receives node embeddings from layer $i-1$ and outputs a new embedding for each node by aggregating the current embeddings of its adjacent neighbors followed by a learnable transformation, as defined below:

$$\mathbf{H}_v^{(i)} = \texttt{upd}\left(\texttt{agg}\left(\{\mathbf{H}_u^{(i-1)} : \forall u \in \mathfrak{N}_v\}\right); \Theta^{(i)}\right),$$

where $\mathfrak{N}_v = \{u : \mathbf{A}_{u,v} \neq 0\}$ denotes the set of adjacent nodes to node v (i.e., nodes with outbound edges toward v), and $\mathbf{H}_u^{(i-1)}$ is the embedding of an adjacent node u at layer $i-1$. $\texttt{agg}(\cdot)$, is a (sub)differentiable, permutation invariant aggregator function, such as SUM, MEAN, or MAX. Finally, $\texttt{upd}(\cdot)$ is a learnable function, such as a multi-layer perceptron (MLP), parameterized by $\Theta^{(i)}$ that takes the aggregated vector and outputs the new embedding $\mathbf{H}_v^{(i)}$. For convenience, we define the matrix-based version of $\texttt{agg}(\cdot)$ and $\texttt{upd}(\cdot)$ by stacking the corresponding vectors of all the nodes into a matrix as:

$$\textsc{Agg}(\mathbf{H}, \mathbf{A}) = \left[\texttt{agg}\left(\{\mathbf{H}_u : \forall u \in \mathfrak{N}_v\}\right) : \forall v \in \mathscr{V}\right]^T,$$
$$\textsc{Upd}(\mathbf{M}; \Theta) = \left[\texttt{upd}\left(\mathbf{M}_v; \Theta\right) : \forall v \in \mathscr{V}\right]^T,$$

where we omitted the layer indicator superscripts for simplicity. Initially, we have $\mathbf{H}^{(0)} = \mathbf{X}$ (i.e., node features) as the input to the GNN's first layer. The last layer generates an output embedding vector for each node, which can be used in different ways depending on the downstream task. For node classification, a softmax layer is applied to the final embeddings $\mathbf{H}^{(K)}$ to obtain the posterior class probabilities $\widehat{\mathbf{Y}}$. The illustration of a typical 3-layer GNN is depicted in Figure 4.5.

4.3.2 Differential Privacy

Differential privacy (DP) (Dwork et al., 2006) is the gold standard for formalizing the privacy guarantees of algorithms that process sensitive data. Informally, DP requires that the algorithm's output distribution be roughly the same regardless of the presence of an individual's data in the dataset. As such, an adversary having access to the data of all but the target individual cannot distinguish whether the target's record is among the input data. The formal definition of DP is as follows.

Definition 3 (Differential Privacy (Dwork et al., 2006))**.** *Given $\epsilon > 0$ and $\delta > 0$, a randomized algorithm $\mathscr{A}$ satisfies (ϵ, δ)-differential privacy, if for all possible pairs of adjacent datasets X*

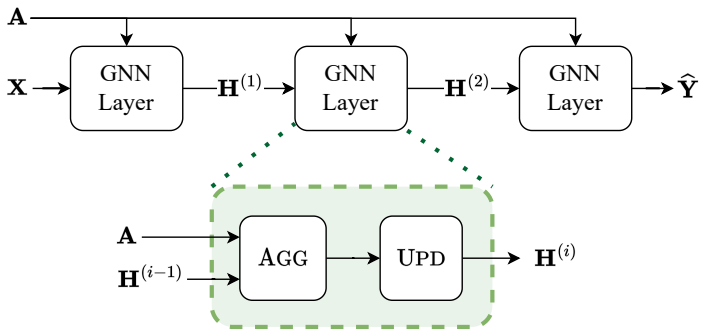


Figure 4.5: Typical 3-layer GNN for node classification. Each layer i takes the adjacency matrix $\mathbf{A}$ and previous layer's node embedding matrix $\mathbf{H}^{(i-1)}$ (initially, node features $\mathbf{X}$), and outputs a new embedding matrix $\mathbf{H}^{(i)}$ (ultimately, predicted class labels $\widehat{\mathbf{Y}}$). Internally, the input embeddings $\mathbf{H}^{(i-1)}$ are aggregated based on the adjacency matrix $\mathbf{A}$, and then fed to a neural network (UPD) to generate new embeddings $\mathbf{H}^{(i)}$.

and X' differing by at most one record, denoted as $X \sim X'$, and for any possible set of outputs $S \subseteq Range(\mathscr{A})$, we have:

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

Here, the parameter ϵ is called the *privacy budget* (or privacy cost) and is used to tune the privacy-utility trade-off of the algorithm: a lower privacy budget leads to stronger privacy guarantees but reduced utility. The parameter δ is informally treated as a failure probability, and is usually chosen to be very small. DP has the following important properties that help us design complex algorithms from simpler ones (Dwork, 2008):

- *Robustness to post-processing:* Any post-processing of the output of an (ϵ,δ)-DP algorithm remains (ϵ,δ)-DP.
- *Sequential composition:* If an (ϵ,δ)-DP algorithm is applied k times on the same data, the result is at most $(k\epsilon, k\delta)$-DP.
- *Parallel composition:* Executing an (ϵ,δ)-DP algorithm on disjoint chunks of data yields an (ϵ,δ)-DP algorithm.

In this chapter, we use an alternative definition of DP, called *Rényi Differential Privacy* (RDP) (Mironov, 2017), which allows obtaining tighter sequential composition results:

Definition 4 (Rényi Differential Privacy (Mironov, 2017))**.** *A randomized algorithm $\mathscr{A}$ is (α,ϵ)-RDP for $\alpha > 1, \epsilon > 0$ if for every adjacent datasets $X \sim X'$, we have $D_\alpha\left(\mathscr{A}(X) \| \mathscr{A}(X')\right) \leq \epsilon$, where $D_\alpha(P\|Q)$ is the Rényi divergence of order α between probability distributions P and Q defined as:*

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

As RDP is a generalization of DP, it can be easily converted back to standard (ϵ, δ)-DP using the following proposition:

Proposition 5. *If $\mathscr{A}$ 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 basic method to achieve RDP is the *Gaussian mechanism*, where Gaussian noise is added to the output of the algorithm we want to make private. Specifically, let $f : \mathscr{X} \to \mathbb{R}^d$ be the non-private algorithm taking a dataset as input and outputting a d-dimensional vector. Let the *sensitivity* of f be the maximum L_2 distance achievable when applying $f(\cdot)$ to adjacent datasets X and X' as $\Delta_f = \max_{X \sim X'} \left\| f(X) - f(X') \right\|_2$. Then, adding Gaussian noise with variance σ^2 to f as $\mathscr{A}(X) = f(X) + \mathscr{N}(\sigma^2 \mathbb{I}_d)$, with $\mathbb{I}_d$ being $d \times d$ identity matrix, yields an (α, ϵ)-RDP algorithm for all $\alpha > 1$ with $\epsilon = \Delta_f^2 \alpha / 2\sigma^2$ (Mironov, 2017).

4.3.3 Problem Definition

Let $\widehat{\mathbf{Y}} = \mathscr{F}(\mathbf{X}, \mathbf{A}; \Theta)$ be a GNN-based node classification model with parameter set Θ that takes node features $\mathbf{X}$ and the graph's adjacency matrix $\mathbf{A}$ as input, and outputs the corresponding predicted labels $\widehat{\mathbf{Y}}$. To learn the model parameters Θ, we minimize a standard classification loss function (e.g., cross-entropy) with respect to Θ as follows:

$$\Theta^\star = \arg\min_{\Theta} \sum_{v \in \mathcal{V}_T} \ell(\widehat{\mathbf{Y}}_v, \mathbf{Y}_v), \tag{4.1}$$

where $\ell(\cdot, \cdot)$ is the loss function, $\mathbf{Y}$ is the ground-truth labels, and $\mathcal{V}_T \subseteq \mathcal{V}$ is the set of labeled training nodes. After training, in the transductive setting, the learned GNN is used to infer the labels of unlabeled nodes in $\mathscr{G}$:

$$\widehat{\mathbf{Y}} = \mathscr{F}(\mathbf{X}, \mathbf{A}; \Theta^\star), \tag{4.2}$$

Otherwise, in the inductive setting, a new graph dataset $\mathscr{G}_{test}$ is given to the learned GNN for label inference.

The goal of this chapter is to preserve the privacy of graph datasets for both the training step (Eq. 4.1) and the inference step (Eq. 4.2) using differential privacy. Note that preserving privacy in the inference step is critical as the adjacency information is still used in this step for obtaining the predicted labels.

However, as graph datasets are different from standard tabular datasets due to the existence of links between data records, one needs to adapt the definition of DP to graphs. As the semantic interpretation of DP relies on the definition of adjacent datasets, we first define two different notions of adjacency in graphs, namely edge-level and node-level adjacent graph datasets (Hay et al., 2009):

Definition 5 (Edge-level adjacent graphs)**.** *Two graphs $\mathscr{G}$ and $\mathscr{G}'$ are edge-level adjacent if one can be obtained by removing a single edge from the other. Therefore, $\mathscr{G}$ and $\mathscr{G}'$ differ by at most one edge.*

Definition 6 (Node-level adjacent graphs)**.** *Two graphs $\mathcal{G}$ and $\mathcal{G}'$ are node-level adjacent if one can be obtained by removing a single node (with its features, labels, and all attached edges) from the other. Therefore, $\mathcal{G}$ and $\mathcal{G}'$ differ by at most one node.*

Accordingly, the definition of edge-level and node-level DP follows from the above definitions: an algorithm $\mathscr{A}$ is edge-level (respectively, node-level) (ϵ, δ)-DP if for every two edge-level (respectively, node-level) adjacent graph datasets $\mathcal{G}$ and $\mathcal{G}'$ and any set of outputs $S \subseteq Range(\mathscr{A})$, we have $\Pr[\mathscr{A}(\mathcal{G}) \in S] \leq e^{\epsilon} \Pr[\mathscr{A}(\mathcal{G}') \in S] + \delta$.

Intuitively, edge-level DP protects edges (which could represent connections between people), while node-level DP protects nodes together with their adjacent edges (i.e., all information pertaining to an individual, including features, labels, and connections).

4.4 Proposed Method: GAP

In this section, we explain our proposed differentially private method, called GNN with Aggregation Perturbation (GAP), which guarantees both edge-level and node-level privacy for training and inference on sensitive graph data.

4.4.1 Overview

As mentioned in Section 4.1, the two primary challenges in the design of private GNNs come from the use of higher-order aggregations and the need to ensure inference privacy. To tackle these challenges, we propose a new architecture for GAP, which is different from the conventional GNN architectures presented in Section 4.3.1. The key distinction is that GAP decouples the graph-based aggregations from the neural network-based transformations, which is similar in spirit to the Inception model and scalable networks (Frasca et al., 2020; Szegedy et al., 2015; F. Wu et al., 2019). As illustrated in Figure 4.3, GAP is composed of the following three components:

(i) *Encoder Module (EM):* This module encodes the input node features into a lower-dimensional representation without using the private graph structure.

(ii) *Aggregation Module (AM):* This module takes the encoded low-dimensional node features and recursively computes private multi-hop aggregations using the *aggregation perturbation* approach, i.e., by adding noise to the output of each aggregation step.

(iii) *Classification Module (CM):* This module takes the privately aggregated node features and predicts the corresponding labels without querying the edges any further.

GAP's privacy mechanism. Our proposed mechanism for preserving the privacy of graph edges in AM is the *aggregation perturbation* approach: we use the Gaussian mechanism to add

stochastic noise to the output of the aggregation function proportional to its sensitivity. This approach is motivated by the fact that perturbing an edge in the input graph can practically be viewed as changing a sample in the neighborhood aggregation function of the edge's destination node. Therefore, by adding an appropriate amount of noise to the aggregation function, we can effectively hide the presence of a single edge, which ensures edge-level privacy, or a group of edges, which is necessary for node-level privacy. To fully guarantee node-level privacy, however, in addition to the edges, we need to also protect node features and labels, which is simply done by training EM and CM using standard DP learning algorithms such as DP-SGD. We discuss this point further in Section 4.5.

Challenges addressed. Our GAP method can benefit from multi-hop aggregations by composing individual noisy aggregation steps. As the sensitivity of a single-step aggregation is easily determined, AM applies the Gaussian mechanism immediately after each aggregation step, avoiding the growing interdependency between node embeddings. GAP also provides inference privacy as the inference of a node relies on the aggregated data from its neighbors, which is privately computed by AM. As the subsequent CM only post-processes these private aggregations, GAP ensures inference-time privacy. This is explained in more details in Section 4.5.

In the rest of this section, we first discuss each of the GAP's components thoroughly and then describe the inference mechanism.

4.4.2 Encoder Module

GAP uses a multi-layer perceptron (MLP) model as an encoder to transform the original node features into an intermediate representation given to AM. The main goal of this module is to reduce the dimensionality of AM's input, as the magnitude of the Gaussian noise injected into the aggregations grows with data dimensionality. Therefore, reducing the dimensionality helps achieve better aggregation utility under DP.

Note that in order to save the privacy budget spent in AM, we do not train the encoder end-to-end with CM. Instead, we attach a linear softmax layer to the encoder MLP for label prediction, and then pre-train this model separately using node features and labels. Specifically, we use the following model:

$$\widehat{\mathbf{Y}} = \text{softmax}\left(\text{MLP}_{\text{enc}}(\mathbf{X}; \Theta_{\text{enc}}) \cdot \mathbf{W}\right), \tag{4.3}$$

where MLP_{enc} is the encoder MLP with parameter set Θ_{enc}, $\mathbf{W}$ is the weight matrix of the linear softmax layer, $\mathbf{X}$ is the original node features, and $\widehat{\mathbf{Y}}$ is the corresponding posterior class probabilities. In order to train this model, we minimize the cross-entropy (or any other

classification-related) loss function $\ell(\cdot,\cdot)$ with respect to the model parameters $\Theta = \{\Theta_{\text{enc}}, \mathbf{W}\}$:

$$\Theta^\star = \arg\min_{\Theta} \sum_{v \in \mathcal{V}_T} \ell(\widehat{\mathbf{Y}}_v, \mathbf{Y}_v), \tag{4.4}$$

where $\mathbf{Y}$ is the ground-truth labels and $\mathcal{V}_T \subseteq \mathcal{V}$ is the set of training nodes. After pre-training, we use the encoder MLP to extract low-dimensional node features, $\mathbf{X}^{(0)}$, for AM:

$$\mathbf{X}^{(0)} = \text{MLP}_{\text{enc}}(\mathbf{X}; \Theta^\star_{\text{enc}}). \tag{4.5}$$

Remark. As will be discussed in Section 4.4.3, this encoder pre-training approach significantly reduces the model's privacy costs as the private aggregations in AM no longer need to be updated with the encoder's parameters. Besides, compared to the original features, this approach provides better node features to AM as the encoded representations incorporate label information as well.

4.4.3 Aggregation Module

The goal of AM is to privately release multi-hop aggregated node features using the aggregation perturbation method. Algorithm 5 presents our mechanism, the Private Multi-hop Aggregation (PMA). It relies on the SUM aggregation function, which is simply equivalent to the multiplication of the adjacency matrix $\mathbf{A}$ by the input feature matrix $\mathbf{X}$, as $\text{AGG}(\mathbf{X}, \mathbf{A}) = \mathbf{A}^T \cdot \mathbf{X}$. The PMA mechanism takes $\check{\mathbf{X}}^{(0)}$, the row-normalized version of the encoder's extracted features as:

$$\check{\mathbf{X}}^{(0)}_v = \mathbf{X}^{(0)}_v / \|\mathbf{X}^{(0)}_v\|_2, \quad \forall v \in \mathcal{V}. \tag{4.6}$$

It then outputs a set of K normalized, privately aggregated node features $\check{\mathbf{X}}^{(1)}$ to $\check{\mathbf{X}}^{(K)}$ corresponding to different hops from 1 to K. Specifically, given $\sigma > 0$, the PMA mechanism performs the following steps to recursively compute and perturb the aggregations in k-th hop from $(k-1)$-th:

1. **Aggregation:** First, we compute k-th non-private aggregations using the normalized aggregations at step $k-1$:
$$\mathbf{X}^{(k)} = \mathbf{A}^T \cdot \check{\mathbf{X}}^{(k-1)}. \tag{4.7}$$

2. **Perturbation:** Next, we perturb the aggregations using the Gaussian mechanism, i.e., by adding noise with variance σ^2 to every row of $\mathbf{X}^{(k)}$ independently:
$$\widetilde{\mathbf{X}}^{(k)}_v = \mathbf{X}^{(k)}_v + \mathcal{N}(\sigma^2 \mathbb{I}), \quad \forall v \in \mathcal{V}. \tag{4.8}$$

3. **Normalization:** Finally, it is essential to bound the effect of each feature vector on the subsequent aggregations. Therefore, we again row-normalize the private aggregated

Algorithm 5: Private Multi-hop Aggregation

Input : Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with adjacency matrix $\mathbf{A}$; initial normalized features $\check{\mathbf{X}}^{(0)}$; max hop K; noise variance σ^2;

Output : Private aggregated node feature matrices $\check{\mathbf{X}}^{(1)}, \ldots, \check{\mathbf{X}}^{(K)}$

for $k \in \{1, \ldots, K\}$ **do**
  $\mathbf{X}^{(k)} \leftarrow \mathbf{A}^T \cdot \check{\mathbf{X}}^{(k-1)}$ // aggregate
  $\widetilde{\mathbf{X}}^{(k)} \leftarrow \mathbf{X}^{(k)} + \mathcal{N}(\sigma^2 \mathbb{I})$ // perturb
  for $v \in \mathcal{V}$ **do**
    $\check{\mathbf{X}}_v^{(k)} \leftarrow \widetilde{\mathbf{X}}_v^{(k)} / ||\widetilde{\mathbf{X}}_v^{(k)}||_2$ // normalize
  end
end
return $\check{\mathbf{X}}^{(1)}, \ldots, \check{\mathbf{X}}^{(K)}$

features, such that the L2-norm of each row is 1:

$$\check{\mathbf{X}}_v^{(k)} = \widetilde{\mathbf{X}}_v^{(k)} / ||\widetilde{\mathbf{X}}_v^{(k)}||_2, \quad \forall v \in \mathcal{V}. \tag{4.9}$$

Remark. The recursive computation of aggregations in the PMA mechanism has one advantage: each aggregation step acts as a denoising mechanism, averaging out the DP noise added in the previous step (to some extent). Therefore, part of the injected noise is dampened by the PMA mechanism itself, leading to better aggregation utility. This noise-reducing effect of GNN aggregations is also observed in prior work (Sajadmanesh & Gatica-Perez, 2021).

Effect of EM. Note that EM plays a critical role in improving AM's privacy-utility trade-off: First, it increases the utility of noisy aggregations by reducing the dimensionality of AM's input, resulting in less noise added to the aggregations. Second, its pre-training strategy makes AM agnostic to model training, which remarkably reduces the total privacy costs as the PMA mechanism is called only once and its output is cached to be reused for entire training and inference. Technically, this implies that with T training iterations, the Gaussian mechanism is composed only K times, which would otherwise be KT in the case of end-to-end training. Since K is small ($1 \leq K \leq 5$) compared to T (in the order of hundreds), this leads to a substantial reduction in the privacy budget.

4.4.4 Classification Module

Given the list of private aggregated features $\{\check{\mathbf{X}}^{(0)}, \ldots, \check{\mathbf{X}}^{(K)}\}$ provided by AM, the goal of CM is to predict node labels without further relying on the graph edges. To this end, for each $k \in \{0, 1, \ldots, K\}$, we first obtain the k-hop representation $\mathbf{H}^{(k)}$ using a corresponding base MLP, denoted as $\text{MLP}_{\text{base}}^{(k)}$:

$$\mathbf{H}^{(k)} = \text{MLP}_{\text{base}}^{(k)}(\check{\mathbf{X}}^{(k)}; \Theta_{\text{base}}^{(k)}), \tag{4.10}$$

where $\Theta_{\text{base}}^{(k)}$ is the parameters of $\text{MLP}_{\text{base}}^{(k)}$. Next, we combine these representations to get an integrated node embedding $\mathbf{H}$:

$$\mathbf{H} = \textsc{Combine}\left(\{\mathbf{H}^{(0)}, \mathbf{H}^{(1)}, \ldots, \mathbf{H}^{(K)}\}; \Theta_{\text{comb}}\right), \tag{4.11}$$

where COMBINE is any differentiable combination strategy, with common choices being summation, concatenation, or attention, potentially with parameter set Θ_{comb}. Finally, we feed the integrated representation into a head MLP, denoted as MLP_{head}, to get posterior class probabilities for the nodes:

$$\widehat{\mathbf{Y}} = \text{MLP}_{\text{head}}(\mathbf{H}; \Theta_{\text{head}}), \tag{4.12}$$

where Θ_{head} denotes the parameters of MLP_{head}. To train CM, we minimize a similar loss function as Eq. 4.4 but with respect to CM's parameters: $\Theta = \{\Theta_{\text{base}}^{(0)}, \ldots, \Theta_{\text{base}}^{(K)}, \Theta_{\text{comb}}, \Theta_{\text{head}}\}$. The overall training procedure of GAP is presented in Algorithm 6.

Remark. CM independently processes the information encoded in the graph-agnostic node features $\check{\mathbf{X}}^{(0)}$ and the private, graph-based aggregated features $\check{\mathbf{X}}^{(1)}$ to $\check{\mathbf{X}}^{(K)}$, combining them together to get an integrated node representation. Therefore, even if the DP noise overwhelms the signal in the higher-level aggregations, the information in the lower-level aggregations and/or the graph-agnostic features is still preserved and exploited for classification. As a result, regardless of the privacy budget, GAP is expected to always perform on par or better than pure MLP-based models that do not rely on the graph structure. We will empirically demonstrate this point in our experiments.

4.4.5 Inference Mechanism

GAP is compatible with both the transductive and the inductive inference, as discussed below.

Transductive setting. In this setting, both training and inference are conducted on the same graph, but using different nodes for training and inference steps (Figure 4.4a). As the entire graph is available at training time, AM computes the private aggregations of all the nodes, including both training and test ones. Therefore, at inference time, we only give the cached aggregations of the test nodes to the trained CM to predict their labels.

Inductive setting. Here, we use a new graph for inference different from the one used for training (Figure 4.4b). In this case, we first extract low-dimensional node features for the new graph using the pre-trained encoder and then feed them to AM to obtain the private aggregations. Finally, we input the private aggregations to the trained CM to get the node labels.

Algorithm 6: GAP Training

Input : Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with adjacency matrix $\mathbf{A}$; node features $\mathbf{X}$; node labels $\mathbf{Y}$; max hop K; noise variance σ^2;

Output : Trained model parameters $\{\Theta^\star{}_{\text{enc}}, \Theta^{\star\,(0)}_{\text{base}}, \ldots, \Theta^{\star\,(K)}_{\text{base}}, \Theta^\star{}_{\text{comb}}, \Theta^\star{}_{\text{head}}\}$;

Pre-train EM (Eq. 4.3) to obtain $\Theta^\star{}_{\text{enc}}$.
Use the pre-trained encoder (Eq. 4.5) to obtain encoded features $\mathbf{X}^{(0)}$.
Row-normalize the encoded features (Eq. 4.6) to obtain $\breve{\mathbf{X}}^{(0)}$.
Use Algorithm 5 to obtain private aggregations $\breve{\mathbf{X}}^{(1)}, \ldots, \breve{\mathbf{X}}^{(K)}$.
Train CM (Eq. 4.10-Eq. 4.12) to get $\Theta^{\star\,(0)}_{\text{base}}, \ldots, \Theta^{\star\,(K)}_{\text{base}}, \Theta^\star{}_{\text{comb}}, \Theta^\star{}_{\text{head}}$.
return $\{\Theta^\star{}_{\text{enc}}, \Theta^{\star\,(0)}_{\text{base}}, \ldots, \Theta^{\star\,(K)}_{\text{base}}, \Theta^\star{}_{\text{comb}}, \Theta^\star{}_{\text{head}}\}$

Algorithm 7: GAP Transductive Inference

Input : Model parameters $\Theta^{\star\,(0)}_{\text{base}}, \ldots, \Theta^{\star\,(K)}_{\text{base}}, \Theta^\star{}_{\text{comb}}, \Theta^\star{}_{\text{head}}$ trained over $\mathcal{G} = (\mathcal{V}, \mathcal{E})$; Cached private aggregations $\breve{\mathbf{X}}^{(0)}, \ldots, \breve{\mathbf{X}}^{(K)}$; Set of query nodes $\mathcal{Q} \subseteq \mathcal{V}$;

Output : Class probabilities $\widehat{\mathbf{Y}}_v$ for all query nodes $v \in \mathcal{Q}$;

for all $k \in \{0, 1, \ldots, K\}$ **do in parallel**
  $\breve{\mathbf{X}}^{(k)}_{\mathcal{Q}} \leftarrow [\breve{\mathbf{X}}^{(k)}_v : \forall\, v \in \mathcal{Q}]^T$
  $\mathbf{H}^{(k)}\mathcal{Q} \leftarrow \text{MLP}^{(k)}_{\text{base}}(\breve{\mathbf{X}}^{(k)}_{\mathcal{Q}}; \Theta^{\star\,(k)}_{\text{base}})$
end
$\mathbf{H}_{\mathcal{Q}} \leftarrow \textsc{Combine}\left(\{\mathbf{H}^{(0)}_{\mathcal{Q}}, \mathbf{H}^{(1)}_{\mathcal{Q}}, \ldots, \mathbf{H}^{(K)}_{\mathcal{Q}}\}; \Theta^\star{}_{\text{comb}}\right)$
$\widehat{\mathbf{Y}}_{\mathcal{Q}} \leftarrow \text{MLP}_{\text{head}}(\mathbf{H}_{\mathcal{Q}}; \Theta^\star{}_{\text{head}})$
return $\widehat{\mathbf{Y}}_{\mathcal{Q}}$

4.5 Privacy Analysis

4.5.1 Edge-Level Privacy

In the following, we provide a formal analysis of GAP's edge-level privacy guarantees at training and inference stages.

Training privacy. The following arguments establish the DP guarantees of the PMA mechanism and the GAP training algorithm:

Theorem 2. *Given the maximum hop $K \geq 1$ and noise variance σ^2, the PMA mechanism presented in Algorithm 5 satisfies edge-level $(\alpha, K\alpha/2\sigma^2)$-RDP for any $\alpha > 1$.*

To prove Theorem 2, we first establish the following lemma.

Lemma 6. *Let $\text{AGG}(\mathbf{X}, \mathbf{A}) = \mathbf{A}^T \cdot \mathbf{X}$ be the summation aggregation function. Assume that the input feature matrix $\mathbf{X}$ is row-normalized, such that $\forall\, v \in \mathcal{V} : \|\mathbf{X}_v\|_2 = 1$. Then, the edge-level sensitivity of the aggregation function is $\Delta_{\text{AGG}} = 1$.*

Algorithm 8: GAP Inductive Inference

Input : Model parameters ${\Theta^\star}_{\text{enc}}$, ${\Theta^\star}^{(0)}_{\text{base}}, \ldots, {\Theta^\star}^{(K)}_{\text{base}}$, ${\Theta^\star}_{\text{comb}}$, ${\Theta^\star}_{\text{head}}$ trained over $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with noise variance σ^2; Test graph $\mathcal{G}_\mathcal{T} = (\mathcal{V}_\mathcal{T}, \mathcal{E}_\mathcal{T})$ with adjacency matrix $\mathbf{A}_\mathcal{T}$ and node features $\mathbf{X}_\mathcal{T}$;

Output : Class probabilities $\widehat{\mathbf{Y}}_\mathcal{T}$

Use the pre-trained encoder (Eq. 4.5) with $\mathbf{X} = \mathbf{X}_\mathcal{T}$ and $\Theta_{\text{enc}} = {\Theta^\star}_{\text{enc}}$ to obtain encoded features $\mathbf{X}^{(0)}_\mathcal{T}$.

Row-normalize the encoded features (Eq. 4.6) to obtain $\check{\mathbf{X}}^{(0)}_\mathcal{T}$.

Apply the PMA mechanism (Algorithm 5) on $\mathbf{A}_\mathcal{T}$ and $\mathbf{X}^{(0)}_\mathcal{T}$ with K hops and σ^2 noise to obtain private aggregations $\check{\mathbf{X}}^{(1)}_\mathcal{T}, \ldots, \check{\mathbf{X}}^{(K)}_\mathcal{T}$.

for all $k \in \{0, 1, \ldots, K\}$ **do in parallel**

  $\mathbf{H}^{(k)}_\mathcal{T} \leftarrow \text{MLP}^{(k)}_{\text{base}}(\check{\mathbf{X}}^{(k)}_\mathcal{T}; {\Theta^\star}^{(k)}_{\text{base}})$

end

$\mathbf{H}_\mathcal{T} \leftarrow \textsc{Combine}\left(\{\mathbf{H}^{(0)}_\mathcal{T}, \mathbf{H}^{(1)}_\mathcal{T}, \ldots, \mathbf{H}^{(K)}_\mathcal{T}\}; {\Theta^\star}_{\text{comb}}\right)$

$\widehat{\mathbf{Y}}_\mathcal{T} \leftarrow \text{MLP}_{\text{head}}(\mathbf{H}_\mathcal{T}; {\Theta^\star}_{\text{head}})$

return $\widehat{\mathbf{Y}}_\mathcal{T}$

Proof. Let $\mathbf{A}$ and $\mathbf{A}'$ be the adjacency matrices of two arbitrary edge-level adjacent graphs. Therefore, there exist two nodes u and v such that:

$$\begin{cases} \mathbf{A}'_{i,j} \neq \mathbf{A}_{i,j}, & \text{if } i = u \text{ and } j = v, \\ \mathbf{A}'_{i,j} = \mathbf{A}_{i,j}, & \text{otherwise.} \end{cases} \tag{4.13}$$

Without loss of generality, we can assume that $\mathbf{A}_{v,u} = 1$ and $\mathbf{A}'_{v,u} = 0$. The goal is to bound the following quantity:

$$\|\textsc{Agg}(\mathbf{X}, \mathbf{A}) - \textsc{Agg}(\mathbf{X}, \mathbf{A}')\|_F.$$

Let $\mathbf{M} = \textsc{Agg}(\mathbf{X}, \mathbf{A})$ be the aggregation function output on $\mathbf{A}$, and

$$\mathbf{M}_i = \sum_{j=1}^{N} \mathbf{A}_{j,i} \mathbf{X}_j,$$

be the i-th row of $\mathbf{M}$ corresponding to the aggregated vector for the i-th node. Analogously, let $\mathbf{M}' = \textsc{Agg}(\mathbf{X}, \mathbf{A}')$. Then:

$$\begin{aligned} \|\textsc{Agg}(\mathbf{X}, \mathbf{A}) - \textsc{Agg}(\mathbf{X}, \mathbf{A}')\|_F &= \|\mathbf{M} - \mathbf{M}'\|_F \\ &= \Big(\sum_{i=1}^{N} \|\mathbf{M}_i - \mathbf{M}'_i\|_2^2\Big)^{1/2} \\ &= \left(\sum_{i=1}^{N} \Big\| \sum_{j=1}^{N} (\mathbf{A}_{j,i}\mathbf{X}_j - \mathbf{A}'_{j,i}\mathbf{X}_j) \Big\|_2^2\right)^{1/2} \\ &= \left(\|\mathbf{A}_{v,u}\mathbf{X}_v - \mathbf{A}'_{v,u}\mathbf{X}_v\|_2^2\right)^{1/2} \end{aligned}$$

$$
\begin{aligned}
&= \|(\mathbf{A}_{v,u} - \mathbf{A}'_{v,u})\mathbf{X}_v\|_2 \\
&= \|\mathbf{X}_v\|_2 \\
&= 1,
\end{aligned}
$$

which concludes the proof. □

We can now prove Theorem 2.

Proof. The PMA mechanism applies the Gaussian mechanism on the output of the summation aggregation function $\text{AGG}(\mathbf{X}, \mathbf{A}) = \mathbf{A}^T \cdot \mathbf{X}$. Based on Lemma 6, the edge-level sensitivity of $\text{AGG}(\cdot)$ is 1. Therefore, according to Corollary 3 of Mironov (2017), each individual application of the Gaussian mechanism is $(\alpha, \alpha/2\sigma^2)$-RDP. As PMA can be seen as an adaptive composition of K such mechanisms, based on Proposition 1 of Mironov (2017), the total privacy cost is $(\alpha, K\alpha/2\sigma^2)$-RDP. □

Proposition 6. *For any $\delta \in (0,1)$, maximum hop $K \geq 1$, and noise variance σ^2, Algorithm 6 satisfies edge-level (ϵ, δ)-DP with:*

$$
\epsilon = \frac{K}{2\sigma^2} + \frac{\sqrt{2K\log(1/\delta)}}{\sigma}
$$

Proof. Under edge-level DP, only the adjacency information is protected. In Algorithm 6, the only step where the graph's adjacency is used is the application of the PMA mechanism (step 4), which according to Theorem 2 is $(\alpha, K\alpha/2\sigma^2)$-RDP. Since EM does not use the graph's edges and the classification module only post-process the private aggregated features without accessing the edges again, the total privacy cost remains $(\alpha, K\alpha/2\sigma^2)$-RDP. Therefore, according to Proposition 5 it 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 $\epsilon = \frac{K}{2\sigma^2} + \sqrt{2K\log(1/\delta)}/\sigma$. □

Proposition 6 shows that the privacy cost grows with the number of hops (K), but is independent of the number of training steps thanks to our GAP architecture.

Inference privacy. A major advantage of GAP is that querying the model at inference time preserves DP *without consuming additional privacy budget*. This is true for both the transductive and the inductive settings:

- *Transductive setting:* In this setting, the inference is performed by feeding the privately trained CM with the cached aggregations of the test nodes, which have already been computed privately at training time. As this computation does not query the private graph structure and only post-processes the previous DP operations, due to the robustness of DP to post-processing, GAP provides inference privacy with no additional cost.

- *Inductive setting:* In this case, first the new graph's node features are given to the encoder to obtain low-dimensional features, which are fed to AM to compute private aggregations. Then, the private aggregations are given to CM to obtain the final predictions. The only part where the private graph structure is queried is the AM, in which the PMA mechanism is applied to the new graph data, and thus the output is private. Furthermore, since the training and test graphs are disjoint, this application of the PMA mechanism is subject to the parallel composition of differentially private mechanisms, and thus it does not increase the privacy costs beyond that of training's. The other parts, the encoder and CM, perform graph-agnostic computations and only post-process previous DP outputs, leading to GAP ensuring inference privacy without extra privacy costs.

4.5.2 Node-Level Privacy

Equipped with aggregation perturbation, the proposed GAP architecture guarantees edge-level privacy by default. However, it is readily extensible to provide node-level privacy guarantees as well, providing that we have bounded-degree graphs, i.e., the degree of each node should be bounded above by a constant D. This allows to bound the sensitivity of the aggregation function in the PMA mechanism when adding/removing a node, as in this case each node can influence at most D other nodes. If the input graph has nodes with very high degrees, we can use neighbor sampling (as proposed in (Daigavane et al., 2021)) to randomly sample at most D neighbors per node.

For bounded-degree graphs, adding or removing a node corresponds (in the worst case) to adding or removing D edges. Therefore, our PMA mechanism also ensures node-level privacy, albeit with increased privacy costs compared to the edge-level setting (see Theorem 3 below).

However, since the node features and labels are also private under node-level DP, both EM and CM need to be trained privately as they access node features/labels. To this end, we can simply use standard DP-SGD (Abadi et al., 2016) or any other differentially private learning algorithm for pre-training the encoder as well as training CM with DP. In other words, steps 1 and 5 of Algorithm 6 must be done with DP instead of regular non-private training. This way, since each of the three GAP modules become node-level private, the entire GAP model, as an adaptive composition of several node-level private mechanisms, satisfies node-level DP. The formal node-level privacy analysis of GAP's training and inference is provided below.

Training privacy. The node-level privacy guarantees of the PMA mechanism and the GAP training algorithm are as follows:

Theorem 3. *Given the maximum degree $D \geq 1$, maximum hop $K \geq 1$, and noise variance σ^2, Algorithm 5 (PMA mechanism) satisfies node-level $(\alpha, {}^{DK\alpha}/_{2\sigma^2})$-RDP for any $\alpha > 1$.*

We first prove Lemma 7 and Lemma 8, and then prove Theorem 3.

Lemma 7. *Given any graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, let*

$$agg(\{\mathbf{X}_u : \forall u \in \mathfrak{N}_v\}) = \sum_{u \in \mathfrak{N}_v} \mathbf{X}_u$$

be the summation aggregation function over the neighborhood $\mathfrak{N}_v$ of any arbitrary node $v \in \mathcal{V}$. Assume that the input feature matrix $\mathbf{X}$ is row-normalized, such that $\forall v \in \mathcal{V} : \|\mathbf{X}_v\|_2 = 1$. Then, the node-level sensitivity of $agg(.)$ is $\Delta_{agg} = 1$.

Proof. Consider a node-level adjacent graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}', \mathbf{X}')$ formed by adding a single node q to $\mathcal{G}$. Hence, we have $\mathcal{V}' = \mathcal{V} \cup \{q\}$, and $\mathbf{X}'_v = \mathbf{X}_v$ for every node $v \in \mathcal{V}$. Let $\mathbf{A}$ and $\mathbf{A}'$ be the adjacency matrices of $\mathcal{G}$ and $\mathcal{G}'$ respectively. The goal is to bound the following:

$$\|\mathsf{agg}(\{\mathbf{X}_u : \forall u \in \mathfrak{N}_v\}) - \mathsf{agg}\left(\{\mathbf{X}'_u : \forall u \in \mathfrak{N}'_v\}\right)\|_2 \leq 1. \tag{4.14}$$

where $\mathfrak{N}_v = \{u : \mathbf{A}_{u,v} = 1\}$ and $\mathfrak{N}'_v = \{u : \mathbf{A}'_{u,v} = 1\}$ are the adjacent nodes to v in $\mathcal{G}$ and $\mathcal{G}'$, respectively. Fixing any arbitrary node $v \in \mathcal{V}$, we have the following two cases:

1. If $q \in \mathfrak{N}'_v$, then we have $\mathfrak{N}_v = \mathfrak{N}'_v \setminus \{q\}$. Therefore:

$$\begin{aligned}
&\|\mathsf{agg}(\{\mathbf{X}_u : \forall u \in \mathfrak{N}_v\}) - \mathsf{agg}\left(\{\mathbf{X}'_u : \forall u \in \mathfrak{N}'_v\}\right)\|_2 \\
&\qquad = \|\sum_{u \in \mathfrak{N}_v} \mathbf{X}_u - \sum_{u \in \mathfrak{N}'_v} \mathbf{X}'_u\|_2 \\
&\qquad = \|\mathbf{X}_q\|_2 = 1.
\end{aligned}$$

2. If $q \notin \mathfrak{N}'_v$, then we have $\mathfrak{N}_v = \mathfrak{N}'_v$. Therefore:

$$\begin{aligned}
&\|\mathsf{agg}(\{\mathbf{X}_u : \forall u \in \mathfrak{N}_v\}) - \mathsf{agg}\left(\{\mathbf{X}'_u : \forall u \in \mathfrak{N}'_v\}\right)\|_2 \\
&\qquad = \|\sum_{u \in \mathfrak{N}_v} \mathbf{X}_u - \sum_{u \in \mathfrak{N}'_v} \mathbf{X}'_u\|_2 = 0.
\end{aligned}$$

Eq. 4.14 follows from the above two cases. □

Lemma 8. *Given any graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ with adjacency matrix $\mathbf{A}$ and maximum degree bounded above by some constant $D > 0$, assume that the feature matrix $\mathbf{X}$ is row-normalized, such that $\forall v \in \mathcal{V} : \|\mathbf{X}_v\|_2 = 1$. Let $agg(\{\mathbf{X}_u : \forall u \in \mathfrak{N}_v\}) = \sum_{u \in \mathfrak{N}_v} \mathbf{X}_u$ be the summation aggregation function over the neighborhood $\mathfrak{N}_v$ of any arbitrary node $v \in \mathcal{V}$, and $\widetilde{\mathrm{AGG}}(\mathbf{X}, \mathbf{A})$ be a noisy aggregation mechanism which applies the Gaussian mechanism independently on the aggregated vector of every individual node as:*

$$\widetilde{\mathrm{AGG}}(\mathbf{X}, \mathbf{A}) = \left[agg(\{\mathbf{X}_u : \forall u \in \mathfrak{N}_v\}) + \mathcal{N}(\sigma^2 \mathbb{I}) : \forall v \in \mathcal{V}\right]^T.$$

Then $\widetilde{\mathrm{AGG}}(.)$ is $(\alpha, D\alpha/2\sigma^2)$-RDP.

Proof. According to Lemma 7, the node-level sensitivity of $\texttt{agg}(\{\mathbf{X}_u : \forall u \in \mathfrak{N}_v\})$ is 1, and thus each individual noisy aggregation query is $(\alpha, \alpha/2\sigma^2)$-RDP. Although $\widetilde{\text{AGG}}$ is composed of $N = |\mathcal{V}|$ such queries in total (one noisy aggregation per node), as $\mathcal{G}$'s maximum degree is bounded above by D, the embedding $\mathbf{X}_u$ of each node u only contributes to maximum D out of N queries. As these N queries are chosen non-adaptively and the noise of the Gaussian mechanism is independently drawn for each query, the maximum privacy cost of $\widetilde{\text{AGG}}(.)$ is equivalent to D compositions of $(\alpha, \alpha/2\sigma^2)$-RDP mechanisms, which based on Proposition 1 of Mironov (2017) is $(D\alpha, \alpha/2\sigma^2)$-RDP. □

Now, we prove Theorem 3.

Proof. At each step of the PMA mechanism, the Gaussian mechanism is applied on every output row of the summation aggregation function $\text{AGG}(\mathbf{X}, \mathbf{A}) = \mathbf{A}^T \cdot \mathbf{X}$. Based on Lemma 8, this mechanism is $(\alpha, \alpha D/2\sigma^2)$-RDP. As PMA can be seen as an adaptive composition of K such mechanisms, based on Proposition 1 of Mironov (2017), the total privacy cost is $(\alpha, \alpha DK/2\sigma^2)$-RDP. □

Proposition 7. *For any $\alpha > 1$, let encoder pre-training (Step 1 of Algorithm 6) and CM training (Step 5 of Algorithm 6) satisfy $(\alpha, \epsilon_1(\alpha))$-RDP and $(\alpha, \epsilon_5(\alpha))$-RDP, respectively. Then, for any $0 < \delta < 1$, maximum hop $K \geq 1$, maximum degree $D \geq 1$, and noise variance σ^2, Algorithm 6 satisfies node-level (ϵ, δ)-DP with $\epsilon = \epsilon_1(\alpha) + \epsilon_5(\alpha) + DK\alpha/2\sigma^2 + \log(1/\delta)/\alpha - 1$.*

Proof. Under node-level DP, all the information pertaining to an individual node, including its features, label, and edges, are private. The first step of Algorithm 6 privately processes the node features and labels so as to satisfy $(\alpha, \epsilon_1(\alpha))$-RDP. Steps 2 and 3 of the algorithm, however, expose the private node features, but then they are processed by steps 4 and 5, which are $(\alpha, DK\alpha/2\sigma^2)$-RDP (according to Theorem 3) and $(\alpha, \epsilon_5(\alpha))$-RDP, respectively. As a result, Algorithm 6 can be seen as an adaptive composition of an $(\alpha, \epsilon_1(\alpha))$-RDP mechanism, an $(\alpha, DK\alpha/2\sigma^2)$-RDP mechanism, and an $(\alpha, \epsilon_5(\alpha))$-RDP mechanism. Therefore, based on Proposition 1 of Mironov (2017), the total node-level privacy cost of Algorithm 6 is $(\alpha, \epsilon_1(\alpha) + DK\alpha/2\sigma^2 + \epsilon_5(\alpha))$-RDP, which ensures $(\epsilon_1(\alpha) + \epsilon_5(\alpha) + \frac{DK\alpha}{2\sigma^2} + \frac{\log(1/\delta)}{\alpha - 1}, \delta)$-DP based on Proposition 5. □

Note that in Proposition 7, we cannot optimize α in closed form as we do not know the precise form of $\epsilon_1(\alpha)$ and $\epsilon_5(\alpha)$. However, in our experiments, we numerically optimize the choice of α on a per-case basis.

Inference privacy. The arguments stated for edge-level inference privacy also hold for node-level privacy. Note that in the inductive setting, the test graph should also have bounded degree for the node-level inference privacy guarantees to hold.

4.6 Discussion

Choice of aggregation function. In this chapter, we used SUM as the default choice of aggregation function. Although other choices of aggregation functions are also possible, we empirically found that SUM is the most efficient choice to privatize, as its sensitivity does not depend on the size of the aggregation set (i.e., number of neighbors), which is itself a quantity that should be computed privately. For example, the calculation of both MEAN and GCN (Kipf & Welling, 2017) aggregation functions depend on the node degrees, and thus requires additional privacy budget to be spent on perturbing node degrees. In any case, SUM is recognized as one of the most expressive aggregation functions in the GNN literature (Corso et al., 2020; K. Xu et al., 2019).

Normalization instead of clipping. The PMA mechanism uses normalization to bound the effect of each individual feature on the SUM aggregation function. While clipping is more common in the private learning literature (e.g., gradient clipping in DP-SGD (Abadi et al., 2016)), we empirically found that normalization is a better choice for aggregation perturbation: CM is then trained on normalized data, which tends to facilitate learning. Normalizing the node embeddings is actually commonly done in non-private GNNs as well to stabilize training (Hamilton et al., 2017a; You et al., 2020).

Limitations. As the PMA mechanism adds random noise to the aggregation function, its utility naturally depends on the size of the node's aggregation set, i.e., the node's degree. Specifically, with a certain amount of noise, the more inbound neighbors a node has, the more accurate its noisy aggregated vector will be. This implies that graphs with higher average degree per node can tolerate larger noise in the aggregation function, and thus GAP can achieve a better privacy-accuracy trade-off on such graphs. Conversely, GAP's performance will suffer if the average degree of the graph is too low, requiring higher privacy budgets to achieve acceptable accuracy. Note however that this is an expected behavior: nodes with fewer inbound neighbors are more easily influenced by a change in their neighborhood compared to nodes with higher degrees, and thus the privacy of low-degree nodes is harder to preserve than high-degree ones. Furthermore, this limitation is not specific to GAP: it is shared by all DP algorithms, whose performance generally suffer from lack of sufficient data.

Edge-level vs. node-level privacy. While GAP can work in either edge-level or node-level privacy settings, it must be emphasized that the former setting is suitable only for the use cases where the node-level information (e.g, features or labels) is not sensitive or is publicly available (e.g., the vertically partitioned graph setting described in (F. Wu et al., 2021)). Whenever node-level information is private as well (e.g., user profiles in a social network), however, edge-level privacy fails to provide appropriate privacy protection, and thus node-level privacy setting has to be enforced.

Table 4.1: Overview of dataset statistics.

DATASET	NODES	EDGES	DEGREE	FEATURES	CLASSES
FACEBOOK	26,406	2,117,924	62	501	6
REDDIT	116,713	46,233,380	209	602	8
AMAZON	1,790,731	80,966,832	22	100	10

4.7 Experimental Setup

In this section, we conduct extensive experiments to empirically evaluate GAP's privacy-accuracy performance and its resilience under privacy attacks. As GAP's privacy guarantees are the same under both transductive and inductive settings, we only focus on the former, which has also more pertinent use cases (e.g., social networks).

4.7.1 Datasets

We evaluate the proposed method on three publicly available node classification datasets, which are medium to large scale in terms of the number of nodes and edges:

Facebook (Traud et al., 2012). This dataset contains the anonymized Facebook social network between UIUC students collected in September 2005. Nodes represent Facebook users and edges indicate friendship. Each node (user) has the following attributes: student/faculty status, gender, major, minor, and housing status, and the task is to predict the class year of users.

Reddit (Hamilton et al., 2017a). This dataset consist of a set of posts from the Reddit social network, where each node represents a post and an edge indicates if the same user commented on both posts. Node features are extracted based on the embedding of the post contents, and the task is to predict the community (subreddit) that a post belongs to.

Amazon (Chiang et al., 2019). The largest dataset used in this chapter represents Amazon product co-purchasing network, where nodes represent products sold on Amazon and an edge indicates if two products are purchased together. Node features are bag-of-words vectors of the product description followed by PCA, and the task is to predict the category of the products.

We preprocess the datasets by limiting the classes to those having 1k, 10k, and 100k nodes on Facebook, Reddit, and Amazon, respectively. We then randomly split the remaining nodes into training, validation, and test sets with 75/10/15% ratios, respectively. Table 4.1 summarizes the statistics of the datasets after preprocessing.

4.7.2 Competing Methods

Edge-level private methods. The following methods are evaluated under edge-level privacy:

- *GAP-EDP:* Our proposed edge-level DP algorithm.
- *SAGE-EDP:* This is the method of F. Wu et al. (2021) that uses the graph perturbation approach, with the popular GraphSAGE architecture (Hamilton et al., 2017a) as its backbone GNN model.

  We perturb the graph's adjacency matrix using the Asymmetric Randomized Response (ARR) (Imola et al., 2022), which performs better than EDGERAND (F. Wu et al., 2021) by limiting the output sparsity.
- *MLP:* A simple MLP model that does not use the graph edges, and thus provides perfect edge-level privacy ($\epsilon = 0$).

Node-level private methods. We compare the following node-level private algorithms:

- *GAP-NDP:* Our proposed node-level DP approach.
- *SAGE-NDP:* This is the method of Daigavane et al. (2021) that adapts the standard DP-SGD method for 1-layer GNNs, with the same GraphSAGE architecture as its backbone model. Since this method does not inherently ensure inference privacy, as suggested by its authors, we add noise to the aggregation function based on its node-level sensitivity at test time and account for the additional privacy cost.
- *MLP-DP:* Similar to MLP, but trained with DP-SGD so as to provide node-level DP without using the graph edges.

We do not consider the approach of Olatunji, Funke, and Khosla (2021) as it requires public graph data and is thus not directly comparable to the others.

Non-private methods. To quantify the accuracy loss of private approaches, we use the following non-private methods ($\epsilon = \infty$):

- *GAP-∞:* a non-private counterpart of the GAP method, where we do not perturb the aggregations.
- *SAGE-∞:* a non-private GraphSAGE model.

4.7.3 Model Implementation Details

For our GAP models (GAP-EDP, GAP-NDP, and GAP-∞), we set the number of MLP_{enc}, MLP_{base}, and MLP_{head} layers to be 2, 1, and 1, respectively. We use concatenation as the COMBINE

function (Eq. 4.11) and tune the number of hops K in $\{1,2,\ldots,5\}$. For the GraphSAGE models (SAGE-EDP, SAGE-NDP, and SAGE-∞), we use the SUM aggregation function and tune the number of message-passing layers in $\{1,2,\ldots,5\}$, except for SAGE-NDP that only supports one message-passing layer. We use a 2-layer and a 1-layer MLP as preprocessing and post-processing before and after the message-passing layers, respectively. For the MLP baselines (MLP and MLP-DP), we set the number of layers to 3. In addition, for both the GAP-NDP and SAGE-NDP methods, we use randomized neighbor sampling to bound the maximum degree D and search for the best D within $\{100,200,300,400\}$. For all methods, we set the number of hidden units to 16 (including the dimension of GAP's encoded representation) and use the SeLU activation function (Klambauer et al., 2017) at every layer. Batch-normalization is used for all methods except the node-level private ones (GAP-NDP, SAGE-NDP, and MLP-DP), for which batch-normalization is not supported.

4.7.4 Training and Evaluation Details

We train the non-private and edge-level private methods using the Adam optimizer over 100 epochs with full-sized batches. For the node-level private algorithms (GAP-NDP, SAGE-NDP, MLP-DP), we use DP-Adam (Gylberth et al., 2017) with maximum gradient norm set to 1, and train each model for 10 epochs with a batch size of 256, 2048, 4096 on Facebook, Reddit, and Amazon, respectively. For our GAP models (GAP-∞, GAP-EDP, and GAP-NDP), we use the same parameter setting for training both the encoder and classification modules. We train all the methods with a learning rate of 0.01 and repeat each combination of possible hyperparameter values 10 times. We pick the best performing model based on validation accuracy, and report the average test accuracy with 95% confidence interval calculated by bootstrapping with 1000 samples.

4.7.5 Privacy Accounting and Calibration

Privacy budget accounting is done via the Analytical Moments Accountant (Y.-X. Wang et al., 2019). We numerically calibrate the noise scale (i.e., the noise standard deviation σ divided by the sensitivity) of PMA (for GAP-EDP and GAP-NDP), ARR (for SAGE-EDP), DP-SGD (for GAP-NDP, SAGE-NDP, and MLP-DP) and the Gaussian mechanism (for inference privacy in SAGE-NDP) to achieve the desired (ε,δ)-DP. We report results for several values of ε, while δ is set to be smaller than the inverse number of private entities (i.e., edges for edge-level privacy, nodes for node-level privacy). For both GAP-NDP and SAGE-NDP, we use the same noise scale for perturbing the gradients (in DP-SGD) and the aggregations (in PMA and Gaussian mechanisms).

Table 4.2: Test accuracy of different methods on the three datasets. The best performing method in each category — none-private, edge-level DP and node-level DP — is highlighted.

Privacy Level	Method	ϵ	Facebook	Reddit	Amazon
Non-Private	GAP-∞	∞	80.0 ± 0.48	**99.4 ± 0.02**	91.2 ± 0.07
	SAGE-∞	∞	**83.2 ± 0.68**	99.1 ± 0.01	**92.7 ± 0.09**
Edge-Level	GAP-EDP	4	**76.3 ± 0.21**	**98.7 ± 0.03**	**83.8 ± 0.26**
	SAGE-EDP	4	50.4 ± 0.69	84.6 ± 1.63	68.3 ± 0.99
	MLP	0	50.8 ± 0.17	82.4 ± 0.10	71.1 ± 0.18
Node-Level	GAP-NDP	8	**63.2 ± 0.35**	**94.0 ± 0.14**	**77.4 ± 0.07**
	SAGE-NDP	8	37.2 ± 0.96	60.5 ± 1.10	27.5 ± 0.83
	MLP-DP	8	50.2 ± 0.25	81.5 ± 0.12	73.6 ± 0.05

4.7.6 Software and Hardware

All the models are implemented in PyTorch (Paszke et al., 2019) using PyTorch-Geometric (PyG) (Fey & Lenssen, 2019). We use the `autodp` library[2] which implements analytical moments accountant, and utilize Opacus (Yousefpour et al., 2021) for training the node-level private models with differential privacy. Experiments are conducted on Sun Grid Engine with NVIDIA GeForce RTX 3090 and NVIDIA Tesla V100 GPUs, Intel Xeon 6238 CPUs, and 32 GB RAM.

4.8 Results

4.8.1 Trade-offs between Privacy and Accuracy

We first compare the accuracy of our proposed methods against the non-private, edge-level private, and node-level private baselines.

We fix the privacy budget to $\epsilon = 8$ for the node-level private methods and $\epsilon = 4$ for the edge-level private ones

(except for MLP, which does not use the graph structure and thus achieves $\epsilon = 0$). The results are presented in Table 4.2. We observe that in the non-private setting, the proposed GAP architecture is competitive with SAGE, with only a slight decrease in accuracy on Facebook and Amazon. Under both edge-level and node-level privacy settings, however, our proposed methods GAP-EDP and GAP-NDP significantly outperform their competitors.

Particularly, under edge-level privacy, GAP-EDP's accuracy is roughly 26, 14, and 15 points higher than the best competitor over Facebook, Reddit, and Amazon, respectively. Under node-level privacy, our proposed GAP-NDP method outperforms the best performing competitor by

[2] https://github.com/yuxiangw/autodp

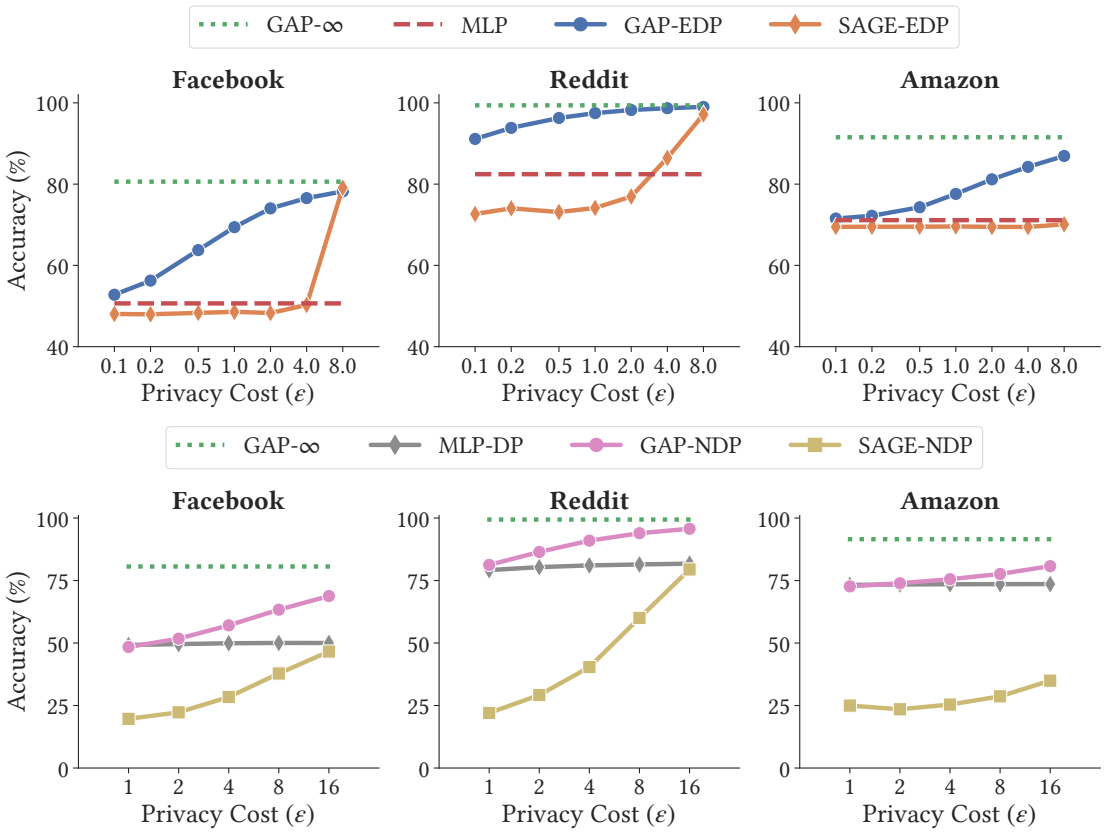


Figure 4.6: Accuracy vs. privacy cost (ϵ) of edge-level private algorithms (top) and node-level private methods (bottom). Note that the y-axis has a different range for each dataset.

approximately 13, 13, and 4 accuracy points, respectively.

Next, to investigate how different methods perform under different privacy budgets,

we vary ϵ from 0.1 to 8 for edge-level private methods and from 1 to 16 for node-level private algorithms

and report the accuracy of the methods under each privacy budget. The result for both edge-level and node-level privacy settings is depicted in Figure 4.6.

Under edge-level privacy (Figure 4.6, left side), we observe that GAP-EDP consistently outperforms its direct competitor, SAGE-EDP, especially at lower privacy costs.

The relative gap between GAP-EDP and SAGE-EDP is influenced by the average degree of the dataset. For example, on Facebook and Reddit with higher average degrees, SAGE-EDP requires a high privacy budget of $\epsilon \geq 8$ to achieve reasonable accuracy, but on Amazon, which has the lowest average degree, it cannot even beat the MLP baseline. In comparison, the accuracy of GAP-EDP approaches the non-private GAP-∞ at much lower privacy budgets, and always performs better than a vanilla MLP. This is because SAGE-EDP perturbs the adjacency matrix,

which is extremely high-dimensional and sparse, while GAP-EDP perturbs the aggregated node embeddings, which has much lower dimensions and is not sparse compared to the adjacency matrix.

The amount of accuracy loss with respect to the non-private method also depends on the average degree of the graph. For example, on Reddit at $\epsilon = 2$, GAP-∞'s accuracy is only 1 point higher than GAP-EDP's, while on Amazon at $\epsilon = 8$, GAP-EDP's accuracy fall behind GAP-∞ by around 5 points. These observations are in line with our discussion of Section 4.6.

We can observe similar trends under node-level privacy (Figure 4.6, right side).

We see that our GAP-NDP method always performs on par or better than the MLP-DP baseline, and also significantly outperforms SAGE-NDP under all the considered privacy budgets. We attribute this to two factors: first, SAGE-NDP is limited to 1-layer models and thus cannot exploit higher-order aggregations; second, the naive noisy aggregation patch for supporting inference privacy severely hurts the performance of SAGE-NDP.

As expected, since the node-level private GAP-NDP hides more information (e.g., node features, labels, and all the adjacent edges to a node) than the edge-level private GAP-EDP, it requires larger privacy budgets to achieve a reasonable accuracy. Still, the accuracy loss with respect to the non-private method is higher in the node-level private method as we have further information loss due to neighborhood sampling (to bound the graph's maximum degree) and gradient clipping (to bound the sensitivity in DP-SGD/Adam).

4.8.2 Resilience Against Privacy Attacks

As mentioned above, the node-level private methods require a higher privacy budget than the edge-level private ones as they attempt to hide much more information. In order to assess the practical implications of choosing rather large privacy budgets (e.g., $\epsilon = 8$ in Table 4.2), we empirically measure the privacy guarantees of GAP-NDP and other node-level private methods by conducting node-level membership inference attack (X. He, Wen, et al., 2021; Olatunji, Nejdl, & Khosla, 2021) as the most relevant adapted privacy attack to GNNs.

Attack overview. The attack is modeled as a binary classification task, where the goal is to infer whether an arbitrary node v is a member of the training set $\mathcal{V}_T$ of the target GNN. The key intuition is that due to overfitting, GNNs give more confident probability scores to training nodes than to test ones, which can be exploited by the attacker to distinguish members of the training set. Having access to a shadow graph dataset coming from the same distribution as the target graph, the attacker first trains a shadow GNN to mimic the behavior of the target GNN, but for which the membership ground truth is known. Then, the attacker trains an attack model over the probability scores of the shadow graph nodes and their corresponding membership labels. Finally, the attacker uses the trained attack model to infer the membership of the target graph nodes.

Table 4.3: Mean AUC of Node Membership Inference Attack.

Dataset	Method	$\epsilon = 1$	$\epsilon = 2$	$\epsilon = 4$	$\epsilon = 8$	$\epsilon = 16$	$\epsilon = \infty$
Facebook	GAP-NDP	50.16	50.25	50.61	51.11	52.66	81.67
	SAGE-NDP	50.25	50.20	50.23	50.17	50.20	62.49
	MLP-DP	50.32	50.72	52.13	53.44	54.77	81.57
Reddit	GAP-NDP	50.04	50.39	51.20	52.23	52.54	54.97
	SAGE-NDP	49.97	49.97	49.95	50.00	49.98	50.05
	MLP-DP	51.25	53.09	55.13	56.72	58.32	71.35
Amazon	GAP-NDP	50.06	50.23	50.54	51.53	51.72	66.68
	SAGE-NDP	49.93	49.93	49.93	49.92	49.97	59.41
	MLP-DP	50.30	50.58	51.43	52.31	53.34	72.97

Attack settings. We follow the TSTF (train on subgraph, test on full graph) approach of Olatunji, Nejdl, and Khosla (2021) for the node-level membership inference attack. Specifically, we consider a strong adversary with access to a shadow graph dataset with 1000 nodes per class, which are sampled uniformly at random from the target dataset. For the shadow model, we use the same architecture and hyperparameters as the target model (described in Section 4.7). Similar to prior work (Olatunji, Nejdl, & Khosla, 2021), we use a 3-layer MLP with 64 hidden units as the attack model, and use the area under the receiver operating characteristic curve (AUC) averaged over 10 runs as the evaluation metric.

Results. Table 4.3 reports the mean AUC of the attack on different node-level private methods trained with the same setting as in Figure 4.6 (right). As we see, the attack is quite effective on the non-private methods ($\epsilon = \infty$), especially on Facebook and Amazon datasets. The success of the attack on each method mainly depends on its generalization gap (the difference between the training and test accuracy): the higher the generalization gap, the more confident the model is on the training nodes and the easier it is to distinguish them from the test nodes. Hence, the lower attack performance on the non-private SAGE method is due to its lower generalization gap compared to the other methods. Nevertheless, for all private GNN methods, we observe that DP with privacy budgets as large as $\epsilon = 16$ can effectively defend against the attack, reducing the AUC to about 50% (random baseline) on all datasets. This result is in line with the work of Jagielski et al. (2020), Jayaraman and Evans (2019), and Nasr et al. (2021), showing that DP with large privacy budgets can still effectively mitigate realistic membership inference attacks.

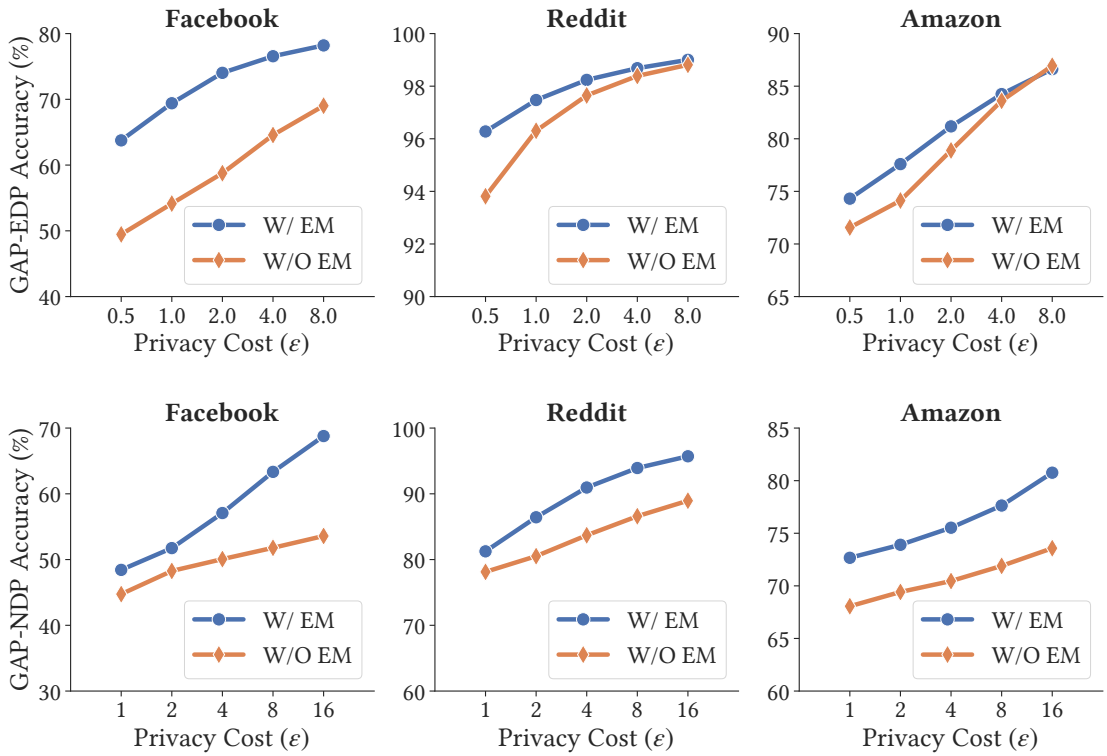


Figure 4.7: Effect of the encoder module (EM) on the accuracy/privacy performance of the edge-level private GAP-EDP (top) and the node-level private GAP-NDP (bottom).

4.8.3 Ablation Studies

Effectiveness of the encoder module (EM). In this experiment, we investigate the effect of EM on the accuracy/privacy performance of the proposed methods, GAP-EDP and GAP-NDP. We compare the case in which EM is used as usual with the case where we remove EM and just input the original node features to the aggregation module. The results under different privacy budgets are given in Figure 4.7. We can observe that in all cases, the accuracy of GAP-EDP and GAP-NDP is higher with EM than without it. For example, leveraging EM results in a gain of around 20, 2, and 5 accuracy points for GAP-EDP with $\epsilon = 1$ on Facebook, Reddit, and Amazon datasets, respectively. GAP-NDP with EM also benefits from a gain of more than 10, 10, and 5 points with $\epsilon = 4$ on Facebook, Reedit, and Amazon datasets, respectively.

As discussed in Section 4.4.2, the improved performance with EM is mainly due to the reduced dimensionality of the aggregation module's input, which leads to adding less noise to the aggregations.

Also, the effect of EM is more significant on GAP-NDP, as the amount of noise injected into the aggregations is generally larger for node-level privacy, hence dimensionality reduction becomes more critical to mitigate the impact of noise.

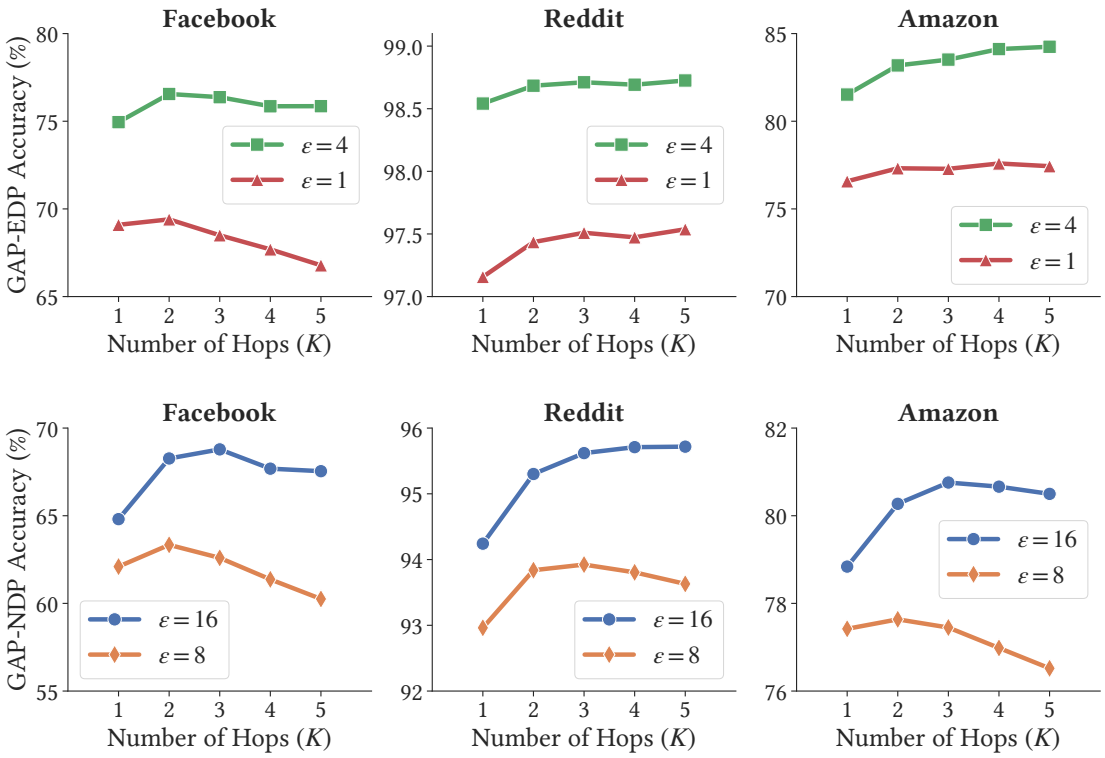


Figure 4.8: Effect of the number of hops K on the accuracy/privacy performance of the edge-level private GAP-EDP (top) and the node-level private GAP-NDP (bottom).

Effect of the number of hops. In this experiment, we investigate how changing the number of hops K affects the accuracy/privacy performance of our proposed methods, GAP-EDP and GAP-NDP. We vary K within $\{1,2,3,4,5\}$ and report the accuracy under different privacy budgets: $\epsilon \in \{1,4\}$ for GAP-EDP and $\epsilon \in \{8,16\}$ for GAP-NDP. The result is depicted in Figure 4.8. We observe that both of our methods can effectively benefit from allowing multiple hops, but there is a trade-off in increasing the number of hops. As we increment K, the accuracy of both GAP-EDP and GAP-NDP method increase up to a point and then steady or decrease in almost all cases. The reason is that with a larger K the model is able to utilize information from more distant nodes (all the nodes within the K-hop neighborhood of a node) for prediction, which can increase the final accuracy. However, as more hops are involved, the amount of noise in the aggregations is also increased, which adversely affects the model's accuracy. We can see that with the lower privacy budgets where the noise is more severe, both GAP-EDP and GAP-NDP achieve their peak accuracy at smaller K values. But as the privacy budget increases, the magnitude of the noise is reduced, enabling the models to benefit from larger K values.

Effect of the maximum degree. We now analyze the effect of D on the performance of our node-level private method.

We vary D from 10 to 400 and report GAP-NDP's accuracy under two different privacy budgets

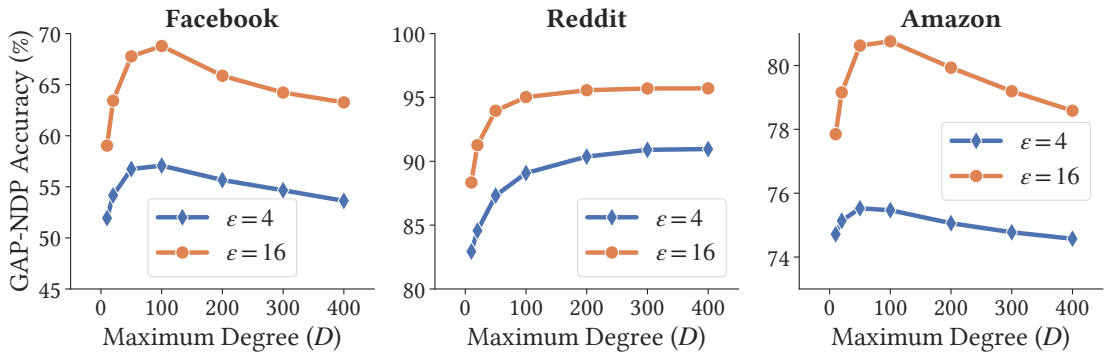


Figure 4.9: Effect of the degree bound D on the accuracy/privacy performance of the node-level private GAP-NDP method.

$\epsilon \in \{4, 16\}$. Figure 4.9 shows that the accuracy keeps growing with D on Reddit (which has a high average degree), while on Facebook and Amazon (lower average degrees) the accuracy increases with D up to a peak point, and drops afterwards.

This is due to the trade-off between having more samples for aggregation and the amount of noise injected: the larger D, the fewer neighbors are excluded from the aggregations (i.e., less information loss), but on the other hand, the larger the sensitivity of the aggregation function, leading to more noise injection. We also observe that the accuracy gain as a result of increasing D gets bigger as the privacy budget is increased from 5 to 20, since a higher privacy budget compensates for the higher sensitivity by reducing the amount of noise.

4.9 Conclusion

In this chapter, we presented GAP, a privacy-preserving GNN architecture that ensures both edge-level and node-level differential privacy for training and inference over sensitive graph data for node-wise learning and inference tasks over graphs. We used aggregation perturbation, where the Gaussian mechanism is applied to the output of the GNN's aggregation function, as a fundamental technique to achieve DP in our approach.

We proposed a new GNN architecture tailored to the specifics of private learning over graphs, aiming to achieve better privacy-accuracy trade-offs while tackling the intricate challenges involved in the design of differentially private GNNs.

More specifically, we decoupled the neighborhood aggregation steps and the learnable transformations into separate aggregation and classification modules, respectively, to avoid spending privacy budget at every training iteration. To further reduce the privacy cost, we proposed to train an encoder module to transform node features into a low-dimensional space without relying on graph adjacency information. Experimental results over real-world graph datasets showed that our approach achieves favorable privacy/accuracy trade-offs and significantly outperforms existing methods. Promising future directions include: (i) investigating robust

aggregation functions that provide specific benefits for private learning; (ii) exploiting the redundancy of information in recursive aggregations to achieve tighter composition when the number of hops K gets large, which might prove useful for specific applications; (iii) extending the framework to other tasks and scenarios, such as link-wise prediction or learning over dynamic graphs; and (iv) conducting an extended theoretical analysis of differentially private GNNs, such as proving utility bounds and characterizing their expressiveness.

5 Progressive Learning on Graphs with Differential Privacy

In the previous chapter, we proposed a privacy-preserving GNN model for learning over graph-structured data with differential privacy guarantees. However, achieving an ideal balance between accuracy and privacy in differntially private GNNs still remains challenging due to the intrinsic structural connectivity of graphs.

In this chapter, 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.

The chapter is organized as follows. Section 5.2 provides an overview of related work on differentially private GNNs. Section 5.3 presents the necessary background and preliminaries on GNNs and differential privacy, which are crucial for understanding the proposed method. In Section 5.4, we formally describe PROGAP and analyze its privacy guarantees. Section 5.5 and Section 5.6 present the experimental setup and results, respectively. Finally, Section 5.7 concludes the chapter and discusses future directions.

The work presented in this chapter is available as a preprint on ArXiv:

- Sajadmanesh, S., & Gatica-Perez, D. (2023). Progap: Progressive graph neural networks with differential privacy guarantees. *arXiv preprint arXiv:2304.08928*

5.1 Introduction

5.1.1 Motivation and Challenges

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 (Ahmedt-Aristizabal et al., 2021; Cheung & Moura, 2020; Gkalelis et al., 2021; W. Jiang & Luo, 2022; Ying et al., 2018). 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 (Corso et al., 2020; Hamilton et al., 2017a; Kipf & Welling, 2017; K. Xu et al., 2019; M. Zhang & Chen, 2018; M. Zhang et al., 2021). 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 (Qiu et al., 2018; J. Wang et al., 2021). 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 (X. He, Jia, et al., 2021a; X. He, Wen, et al., 2021; Olatunji, Nejdl, & Khosla, 2021; F. Wu et al., 2021), 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 (Dwork, 2008; Dwork et al., 2006). 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) (Abadi et al., 2016), infeasible (Ayle et al., 2023; Daigavane et al., 2021; Sajadmanesh et al., 2023). Recently, the *aggregation perturbation* (AP) approach (Sajadmanesh et al., 2023) 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* (Raskhodnikova & Smith, 2016).

The key limitation of AP is its incompatibility with standard GNN architectures due to the high privacy costs it entails (Sajadmanesh et al., 2023). 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. (2023) 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 (Fey et al., 2021), leading to suboptimal accuracy-privacy trade-offs.

5.1.2 Contributions

In the face of these challenges, we present a novel differentially private GNN, called *"**Pro**gressive **G**NN with **A**ggregation **P**erturbation"* (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.

- We propose ProGAP, a novel differentially private GNN framework that combines the aggregation perturbation technique with progressive training to mitigate the high privacy costs associated with the aggregation perturbation method, with a higher expressive power for attaining better accuracy-privacy trade-offs.

- We prove that ProGAP provides similar privacy guarantees as GAP, including edge- and node-level privacy guarantees and zero-cost privacy at inference time.

- We conduct extensive experiments on a variety of node classification datasets and demonstrate that ProGAP achieves state-of-the-art performance under both edge- and node-level privacy in terms of accuracy-privacy trade-off.

5.2 Related Work

Graph neural networks (GNNs) have gained tremendous attention in recent years due to their effectiveness in learning representations of unstructured data such as graphs and networks. Since the introduction of the Graph Convolutional Network (GCN) by Kipf and Welling (Kipf & Welling, 2017) in 2017, numerous GNN variants have been proposed, including Graph Attention Networks (Veličković et al., 2018), GraphSAGE (Hamilton et al., 2017a), Graph Isomorphism Networks (K. Xu et al., 2019), Jumping Knowledge Networks (K. Xu et al., 2018), and Gated Graph Neural Networks (Y. Li et al., 2016), among others. To explore recent developments and current trends in GNNs, we recommend referring to the available surveys (Abadal et al., 2021; Hamilton et al., 2017b; Z. Wu et al., 2020; Z. Zhang, Cui, & Zhu, 2022; Zhou, Cui, et al., 2020).

Recent works have explored the potential for privacy attacks on GNNs and quantified the extent of privacy leakage from publicly released GNN models or node embeddings trained on private graph datasets. For instance, Duddu et al. (2020) conducted a comprehensive study on quantifying privacy leakage of graph embedding algorithms trained on sensitive graph data and proposed various privacy attacks on GNNs. Similarly, Z. Zhang, Chen, et al. (2022) proposed three inference attacks against GNNs, including inferring graph properties and reconstructing graphs with similar statistics to the target graph. Other works, such as X. He, Jia, et al. (2021a), Z. Zhang et al. (2021b), and F. Wu et al. (2021), have also proposed attacks on GNNs, such as link stealing and membership inference attacks. These studies highlight the privacy risks of GNNs trained on sensitive graph data and demonstrate the models' vulnerability to various privacy attacks.

Several recent studies have investigated differential privacy (DP) to provide formal privacy guarantees in various GNN learning settings. For example, Sajadmanesh and Gatica-Perez (2021) 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. (2022) 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 (Kasiviswanathan et al., 2011), which operates under a different problem setting from our method.

Other approaches propose edge-level DP algorithms for GNNs. F. Wu et al. (2021) 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. (2022) 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, Funke, and Khosla (2021) propose the first node-level private GNN by adapting the framework of PATE (Papernot et al., 2016). 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. (2021) 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. (2023) 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 neural 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 (R. Wu et al., 2020), image super-resolution (Y. Wang et al., 2018), image synthesis (Karras et al., 2017), and representation learning (Z. Li et al., 2020). This technique has also been extended to federated learning, mainly to minimize the communication overhead between clients and the central server (Belilovsky et al., 2020; C. He, Li, et al., 2021; H.-P. Wang et al., 2022). However, the potential benefit of progressive learning in DP applications has not been explored yet. In this chapter, we are first to examine the advantages of progressive learning in the context of private GNNs.

5.3 Background

This section begins by presenting key concepts and notations that will be used throughout the chapter. Following this, we formally define the problem of training graph neural networks (GNNs) with differential privacy (DP).

5.3.1 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 (Dwork et al., 2006):

Definition 7. *Given $\epsilon > 0$ and $\delta \in [0,1]$, a randomized algorithm $\mathscr{A}$ satisfies (ϵ, δ)-differential privacy, if for all adjacent datasets $\mathscr{D}$ and $\mathscr{D}'$ differing by at most one record and for all possible*

subsets of $\mathscr{A}$'s outputs $S \subseteq Range(\mathscr{A})$:

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

The value of epsilon, which is often called privacy budget or privacy cost parameter, determines the strength of the privacy guarantee provided by the algorithm, with smaller values of epsilon indicating stronger privacy protection but potentially lower utility of the algorithm. The parameter δ represents the maximum allowable failure probability, i.e., the probability that the algorithm may violate the privacy guarantee, and is usually set to a small value.

The privacy analysis of the proposed method in this chapter is based on *Rényi Differential Privacy* (RDP) (Mironov, 2017), which is an alternative definition of DP that gives tighter sequential composition results. The formal definition of RDP is as follows:

Definition 8 (Rényi Differential Privacy (Mironov, 2017))**.** *Given $\alpha > 1$ and $\epsilon > 0$, a randomized algorithm $\mathscr{A}$ satisfies (α, ϵ)-RDP if for every adjacent datasets X and X', we have:*

$$D_{\alpha}\left(\mathscr{A}(X) \| \mathscr{A}(X')\right) \leq \epsilon, \tag{5.1}$$

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 Mironov (2017), as follows:

Proposition 8 (From RDP to (ϵ, δ)-DP (Mironov, 2017))**.** *If $\mathscr{A}$ is an (α, ϵ)-RDP algorithm, then it also satisfies $(\epsilon + {}^{\log(1/\delta)}/_{\alpha - 1}, \delta)$-DP for any $\delta \in (0, 1)$.*

The guarantee of differential privacy depends on the notion of adjacency between datasets. In the case of tabular datasets, differential privacy defines adjacency between two datasets as being able to obtain one dataset from the other by removing (or replacing) a single record. However, for graph datasets, adjacency needs to be defined differently due to the presence of links between data records. 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 (Raskhodnikova & Smith, 2016). Specifically, an algorithm $\mathscr{A}$ provides edge-/node-level (ϵ, δ)-DP if for every two edge-/node-level adjacent graph datasets $\mathscr{G}$ and $\mathscr{G}'$ and any set of outputs $S \subseteq Range(\mathscr{A})$, we have $\Pr[\mathscr{A}(\mathscr{G}) \in S] \leq e^{\epsilon} \Pr[\mathscr{A}(\mathscr{G}') \in S] + \delta$.

The concepts of edge-level and node-level differential privacy can be intuitively understood as providing privacy protection at different levels of granularity in graph datasets. Edge-

level differential privacy is focused on protecting the privacy of edges, which may represent connections between individuals. In contrast, node-level differential privacy aims to protect the privacy of nodes and their adjacent edges, thus safeguarding all information related to an individual, including their features, labels, and connections.

The difference in granularity between edge-level and node-level DP is crucial because the level of privacy protection needed may depend on the sensitivity of the information being disclosed. For example, protecting only the privacy of individual edges may be sufficient for some applications, while others may require more stringent privacy guarantees that protect the privacy of entire nodes. Our proposed method, which we discuss in detail in Section 5.4, is capable of providing both edge-level and node-level privacy guarantees.

5.3.2 Graph Neural Networks

The goal of GNNs is to learn a vector representation, also known as an embedding, for each node in a given graph. These embeddings are learned by taking into account the initial features of the nodes and the structure of the graph (i.e., its edges). The learned embeddings can be applied to various downstream machine learning tasks, such as node classification and link prediction.

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:

$$\mathbf{X}^{(k)} = \textsc{Upd}\left(\textsc{Agg}(\mathbf{A}, \mathbf{X}^{(k-1)}); \Theta^{(k)}\right), \tag{5.2}$$

where Agg is a differentiable permutation-invariant *neighborhood aggregation function*, such as mean, sum, or max pooling, and Upd denotes a learnable transformation, such as a multi-layer perceptron (MLP), parameterized by $\Theta^{(k)}$ that takes the aggregated embeddings as input and produces a new embedding for each node.

5.3.3 Problem Definition

Consistent with prior work (Daigavane et al., 2021; Sajadmanesh et al., 2023), we focus on the node classification task. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a directed, unweighted graph with a set of nodes $\mathcal{V} = \{v_1, \ldots, v_N\}$ and edges $\mathcal{E}$ represented by an adjacency matrix $\mathbf{A} \in \{0,1\}^{N \times N}$. The graph is associated with a set of node features and ground-truth labels. 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. Node labels are denoted by $\mathbf{Y} \in \{0,1\}^{N \times C}$, where C is the number of classes, and $\mathbf{Y}_i$ is a one-hot vector indicating the label of node v_i. We assume that the node labels are known only for a subset of nodes $\mathcal{V}_L \subset \mathcal{V}$. This reflects the transductive (semi-supervised) learning setting, where the goal

is to predict the labels of the remaining nodes in $\mathcal{V} \setminus \mathcal{V}_L$.

Consider a GNN-based node classification model $\mathscr{M}(\mathbf{A}, \mathbf{X}; \Theta)$ with parameter set Θ that takes the adjacency matrix $\mathbf{A}$ and the node features $\mathbf{X}$, and outputs the corresponding predicted node labels $\widehat{\mathbf{Y}}$:

$$\widehat{\mathbf{Y}} = \mathscr{M}(\mathbf{A}, \mathbf{X}; \Theta). \tag{5.3}$$

We seek to minimize a standard classification loss function $\mathscr{L}$ with respect to the set of model parameters Θ over the labeled nodes $\mathcal{V}_L$:

$$\begin{aligned} \Theta^\star &= \arg\min_{\Theta} \mathscr{L}(\widehat{\mathbf{Y}}, \mathbf{Y}) \\ &= \arg\min_{\Theta} \left(\sum_{v_i \in \mathcal{V}_L} \ell(\widehat{\mathbf{Y}}_i, \mathbf{Y}_i) \right), \end{aligned} \tag{5.4}$$

where $\ell : \mathbb{R}^C \times \mathbb{R}^C \to \mathbb{R}$ is a loss function, such as cross-entropy, and $\Theta^\star$ denotes the optimal set of parameters.

Our goal is to ensure the privacy of $\mathscr{G}$ at both the training (Eq. 5.4) and inference (Eq. 5.3) phases of the model $\mathscr{M}$, 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 (Sajadmanesh et al., 2023).

5.4 Proposed Method

In this section, we present our proposed ProGAP method, which leverages the aggregation perturbation (AP) technique (Sajadmanesh et al., 2023) 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 5.1, and its forward propagation (inference) and training algorithms are presented in Algorithm 9 and Algorithm 10, respectively. In the following, we first describe our method in detail and then analyze its privacy guarantees.

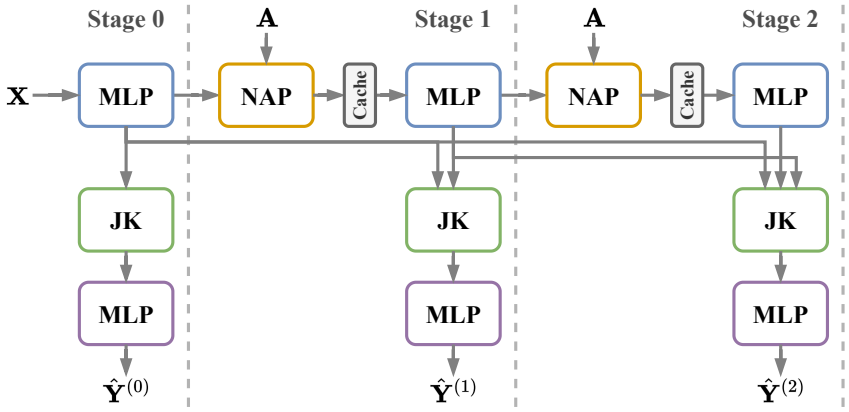


Figure 5.1: An example ProGAP architecture with three stages. MLP and JK represent multi-layer perceptron and Jumping Knowledge (K. Xu et al., 2018) modules, respectively. NAP denotes the normalize-aggregate-perturb module used to ensure the privacy of the adjacency matrix, with its output cached immediately 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.

5.4.1 Model Architecture and Training

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

$$\mathbf{X}^{(0)} = \text{MLP}^{(0)}_{base}\left(\mathbf{X}; \Theta^{(0)}_{base}\right), \tag{5.5}$$

$$\mathbf{X}^{(k)} = \text{MLP}^{(k)}_{base}\left(\text{AGG}(\mathbf{A}, \mathbf{X}^{(k-1)}); \Theta^{(k)}_{base}\right), \quad \forall k \in \{1, \ldots, K\}, \tag{5.6}$$

$$\hat{\mathbf{Y}} = \text{MLP}_{head}\left(\mathbf{X}^{(K)}; \Theta_{head}\right), \tag{5.7}$$

where $\mathbf{X}^{(k)}$ is the node embeddings generated at layer k by $\text{MLP}^{(k)}_{base}$ having parameters $\Theta^{(k)}_{base}$, 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. (2023) and add noise to the output of the aggregation function. Specifically, we replace the original aggregation function AGG in Eq. 5.6 with a *Normalize-Aggregate-Perturb* mechanism defined as:

$$\text{NAP}(\mathbf{A}, \mathbf{X}; \sigma) = \left[\sum_{j=1}^{N} \frac{\mathbf{X}_i}{\|\mathbf{X}_i\|_2} \mathbf{A}_{j,i} + \mathscr{N}(\mathbf{0}, \sigma^2 \mathbf{I}_d) \mid \forall i \in \{1, \ldots, N\} \right], \tag{5.8}$$

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 NAP mechanism row-normalizes the input embeddings to limit the contribution of each node to the aggregated output, then

Algorithm 9: 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 $\mathfrak{P}_s = \bigcup_{k=0}^{s}\{\Theta_{base}^{(k)}\} \cup \{\Theta_{jump}^{(s)}, \Theta_{head}^{(s)}\}$

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

$\widetilde{\mathbf{X}}^{(0)} \leftarrow \mathbf{X}$
for $k \in \{0,\dots,s\}$ **do**
  if $k>0$ *and* $\widetilde{\mathbf{X}}^{(k)}$ *is not cached* **then**
    $\widetilde{\mathbf{X}}^{(k)} \leftarrow \mathrm{NAP}(\mathbf{A},\widetilde{\mathbf{X}}^{(k-1)};\sigma)$
    Cache $\widetilde{\mathbf{X}}^{(k)}$
  end
  $\mathbf{X}^{(k)} \leftarrow \mathrm{MLP}_{base}^{(k)}\left(\widetilde{\mathbf{X}}^{(k)};\Theta_{base}^{(k)}\right)$
end
$\widehat{\mathbf{Y}}^{(s)} \leftarrow \mathrm{MLP}_{head}^{(s)}\left(\mathrm{JK}^{(s)}(\{\mathbf{X}^{(0)},\dots,\mathbf{X}^{(s)}\};\Theta_{jump}^{(s)});\Theta_{head}^{(s)}\right)$
return $\widehat{\mathbf{Y}}^{(s)}$

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:

$$\widetilde{\mathbf{X}}^{(s)} = \mathrm{NAP}\left(\mathbf{A},\mathbf{X}^{(s-1)};\sigma\right), \tag{5.9}$$

$$\mathbf{X}^{(s)} = \mathrm{MLP}_{base}^{(s)}\left(\widetilde{\mathbf{X}}^{(s)};\Theta_{base}^{(s)}\right), \tag{5.10}$$

$$\widehat{\mathbf{Y}}^{(s)} = \mathrm{MLP}_{head}^{(s)}\left(\mathrm{JK}^{(s)}(\bigcup_{k=0}^{s}\{\mathbf{X}^{(k)}\};\Theta_{jump}^{(s)});\Theta_{head}^{(s)}\right), \tag{5.11}$$

where $\widetilde{\mathbf{X}}^{(s)}$ is the noisy aggregate embeddings of $\mathcal{M}_s$, with $\widetilde{\mathbf{X}}^{(0)} = \mathbf{X}$. $\mathrm{JK}^{(s)}$ is a Jumping Knowledge module (K. Xu et al., 2018) with parameters $\Theta_{jump}^{(s)}$ that combines the embeddings generated by submodels $\mathcal{M}_0$ to $\mathcal{M}_s$, and $\mathrm{MLP}_{head}^{(s)}$ is a lightweight, 1-layer head MLP with parameters $\Theta_{head}^{(s)}$ used to train $\mathcal{M}_s$. Finally, $\widehat{\mathbf{Y}}^{(s)}$ 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 the deepest submodel $\mathcal{M}_K$ (which is equivalent to the full model $\mathcal{M}$) as explained by Algorithm 10. For the final inference after training, we simply use the labels predicted by the last submodel $\mathcal{M}_K$, i.e., $\widehat{\mathbf{Y}} = \widehat{\mathbf{Y}}^{(K)}$.

Algorithm 10: 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 $\mathfrak{P}_K^\star$

initialize $\Theta_{base}^{(0)}, \Theta_{jump}^{(0)}, \Theta_{head}^{(0)}$ randomly

$\mathfrak{P}_0 \leftarrow \{\Theta_{base}^{(0)}, \Theta_{jump}^{(0)}, \Theta_{head}^{(0)}\}$

for $s \in \{0, \dots, K\}$ **do**

  $\mathfrak{P}_s^\star \leftarrow \arg\min_{\mathfrak{P}} \mathscr{L}\left(\mathscr{M}_s\left(\mathbf{A}, \mathbf{X}; \sigma, \mathfrak{P}_s\right), \mathbf{Y}\right)$

  if $s < K$ **then**

    initialize $\Theta_{base}^{(s+1)}, \Theta_{jump}^{(s+1)}, \Theta_{head}^{(s+1)}$ randomly

    $\mathfrak{P}_{s+1} \leftarrow \mathfrak{P}_s^\star \cup \{\Theta_{base}^{(s+1)}, \Theta_{jump}^{(s+1)}, \Theta_{head}^{(s+1)}\} \setminus \{\Theta^{\star(s)}_{jump}, \Theta^{\star(s)}_{head}\}$

  end

end

return $\mathfrak{P}_K^\star$

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 $\widetilde{\mathbf{X}}^{(s)}$ computed in the first forward pass of $\mathscr{M}_s$ (via Eq. 5.9) 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 $\widetilde{\mathbf{X}}^{(s)}$ 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* $\mathrm{MLP}_{base}^{(k)}$ *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 5.6, we show that such a progressive training strategy leads to better performance than layerwise training.*

5.4.2 Privacy Analysis

With the following theorem, we show that the proposed training strategy provides edge-level DP:

Theorem 4. *Given the maximum stage* $K \geq 0$ *and noise variance* σ^2*, for any* $\delta \in (0,1)$ *Algorithm 10 satisfies edge-level* (ε, δ)*-DP with* $\varepsilon = \frac{K}{2\sigma^2} + \sqrt{2K\log(1/\delta)}/\sigma$.

Proof. In Algorithm 10, the graph's adjacency is only used when the NAP mechanism is

invoked during the forward propagation of submodels $\mathscr{M}_1$ to $\mathscr{M}_K$. According to Lemma 1 of Sajadmanesh et al. (2023), the edge-level sensitivity of the NAP mechanism is 1, and thus based on Corollary 3 of Mironov (2017), 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 10 can be seen as an adaptive composition of K NAP mechanisms, which based on Proposition 1 of Mironov (2017), is $(\alpha, K\alpha/2\sigma^2)$-RDP. According to Proposition 3 of Mironov (2017), 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 $\epsilon = \frac{K}{2\sigma^2} + \sqrt{2K\log(1/\delta)}/\sigma$. □

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 5 establishes the node-level DP guarantee of ProGAP's training algorithm when combined with DP-SGD:

Theorem 5. *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 $\sigma^2_{AP} > 0$, and noise variance for gradient perturbation $\sigma^2_{GP} > 0$, Algorithm 10 satisfies node-level (ϵ, δ)-DP for any $\delta \in (0,1)$ with:*

$$\epsilon \leq \min_{\alpha>1} \quad \frac{(K+1)T}{\alpha-1} \log\Bigg\{\left(1-\frac{B}{N}\right)^{\alpha-1}\left(\alpha\frac{B}{N}-\frac{B}{N}+1\right) + \binom{\alpha}{2}\left(\frac{B}{N}\right)^2\left(1-\frac{B}{N}\right)^{\alpha-2} e^{\frac{C^2}{\sigma^2_{GP}}} + \sum_{l=3}^{\alpha}\binom{\alpha}{l}\left(1-\frac{B}{N}\right)^{\alpha-l}\left(\frac{B}{N}\right)^{l} e^{(l-1)\left(\frac{C^2 l}{2\sigma^2_{GP}}\right)}\Bigg\} + \frac{DK\alpha}{2\sigma^2_{AP}} + \frac{\log(1/\delta)}{\alpha-1},$$

providing that the optimization in line 4 of Algorithm 10 is done using DP-SGD.

Proof. Algorithm 10 is composed of $K+1$ stages, where each stage $s \in \{1,\ldots,K\}$ starts by computing and perturbing the aggregate embeddings (Eq. 5.9), 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 10 can be seen as an adaptive composition of K NAP mechanisms and $K+1$ DP-SGD algorithms. According to Lemma 3 of Sajadmanesh et al. (2023), the NAP mechanism is node-level $(\alpha, D\alpha/2\sigma^2_{AP})$-RDP. The DP-SGD algorithm itself is a composition of T subsampled Gaussian mechanisms, which according to Theorem 11 of Y. Zhu and Wang

(2019) and Proposition 1 of Mironov (2017) is $(\alpha, \epsilon_{\text{DPSGD}})$-RDP, where:

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

Overall, according to Proposition 1 of Mironov (2017), the composition of K NAP mechanisms and $K+1$ DP-SGD algorithms is $(\alpha, \epsilon_{\text{total}})$-RDP, where:

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

The proof is completed by applying Proposition 3 of Mironov (2017) to the above expression and minimizing the upper bound over $\alpha > 1$:

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

□

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 (Daigavane et al., 2021; Sajadmanesh et al., 2023).

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

Table 5.1: Dataset Statistics

DATASET	# NODES	# EDGES	# FEATURES	# CLASSES	MEDIAN 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
FB-100	1,120,280	86,304,478	537	6	57
WENET	37,576	22,684,206	44	4	286

aggregate matrices $\widetilde{\mathbf{X}}^{(i)}$ corresponding to all the nodes –both training and test ones– are already computed and cached during training and reused for inference (i.e., line 4 and line 5 of Algorithm 9 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.

5.5 Experimental Setup

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

5.5.1 Datasets

We conduct experiments on three real-world datasets that have been used in previous work (Daigavane et al., 2021; Olatunji, Funke, & Khosla, 2021; Sajadmanesh et al., 2023), namely Facebook (Traud et al., 2012), Reddit (Hamilton et al., 2017a), and Amazon (Chiang et al., 2019), and also two new datasets: FB-100 (Traud et al., 2012) and WeNet (Giunchiglia et al., 2021; Meegahapola et al., 2023). 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. FB-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 5.1.

5.5.2 Baselines

We compare ProGAP against the following baselines:

GraphSAGE (Hamilton et al., 2017a). This is one of the most popular GNN models, which we use for non-private performance comparison with our method. Moreover, it serves as the backbone model for the following EdgeRand and DP-GNN baselines. The number of message-passing layers is tuned in the range of 1 to 5 for each dataset. We also use one preprocessing and one postprocessing layer, and equip the model with jumping knowledge modules (K. Xu et al., 2018) to get better performance.

MLP and DP-MLP. We use a simple 3-layer MLP model which does not use any graph structural information and therefore is perfectly edge-level private. DP-MLP is the node-level private variant of MLP, which is trained using the DP version of the Adam optimizer (DP-Adam).

EdgeRand (F. Wu et al., 2021). This edge-level private method directly adds noise to the adjacency matrix. We use the enhanced version of EdgeRand from (Sajadmanesh et al., 2023) that uses the Asymmetric Randomized Response (Imola et al., 2022) with the GraphSAGE model as the backbone GNN.

DP-GNN (Daigavane et al., 2021). This node-level private approach extends the DP-SGD algorithm to GNNs. Note, however, that this method does not support inference privacy, but we nevertheless include it in our comparison. Similar to EdgeRand, we use the GraphSAGEmodel as the backbone GNN for this method as well.

GAP (Sajadmanesh et al., 2023). GAP is the state-of-the-art approach that supports both edge-level and node-level DP. We use GAP's official implementation on GitHub[1] and follow the same experimental setup as reported in the original paper.

We do not include other available differentially private GNN baselines (e.g., (Ayle et al., 2023; Mueller et al., 2022; Olatunji, Nejdl, & Khosla, 2021; Sajadmanesh & Gatica-Perez, 2021)) as they have different problem settings that make them not directly comparable to our method.

5.5.3 Implementation Details

We follow the same experimental setup as GAP (Sajadmanesh et al., 2023), and randomly split the nodes in all the datasets into 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

[1] https://github.com/sisaman/GAP

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 (Klambauer et al., 2017). 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 FB-100, and 400 for Reddit and WeNet. We use DP-Adam (Gylberth et al., 2017) 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, WeNet, Reddit, Amazon, and FB-100, respectively. Finally, we report the average test accuracy over 10 runs with 95% confidence intervals calculated by bootstrapping with 1000 samples.

5.5.4 Software and Hardware

We use PyTorch Geometric (Fey & Lenssen, 2019) for implementing the models, `autodp`[2] for privacy accounting, and Opacus (Yousefpour et al., 2021) for DP training. We run all the experiments on an HPC cluster with NVIDIA Tesla V100 and GeForce RTX 3090 GPUs with maximum 32GB memory.

5.6 Results and Discussion

5.6.1 Accuracy-Privacy Trade-off

Table 5.2 presents the test accuracy of ProGAP against other baselines 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 ± 95% confidence interval. We observe that ProGAP outperforms other approaches most of the time, and often by a substantial margin. In the non-private setting, ProGAP performs comparably with GraphSAGE, but achieves higher test accuracies than GAP on all datasets except Reddit, where GAP performs only slightly better. This shows that our ProGAP method is also a strong predictor in the non-private learning setting.

Moving to edge-level privacy, ProGAP consistently outperforms other approaches across all datasets, with the largest performance gap of 10.4% accuracy points compared to GAP observed on FB-100. Under node-level DP, ProGAP is still superior to the other baselines across all datasets except Reddit, where DP-GNN achieves a slightly higher accuracy. Note, however, that DP-GNN only guarantees node-level privacy for the model parameters and fails to provide inference privacy, while our ProGAP method provides both training and inference privacy guarantees. Compared to GAP, the largest margin in this category is also observed on

[2] https://github.com/yuxiangw/autodp

Table 5.2: Comparison of Experimental Results (Mean Accuracy ± 95% CI)

Privacy Level	Method	ϵ	Facebook	Reddit	Amazon	FB-100	WeNet
Non-Private	GraphSAGE	∞	**84.7 ± 0.09**	99.4 ± 0.01	93.2 ± 0.07	74.0 ± 0.80	71.6 ± 0.54
	GAP	∞	80.5 ± 0.42	**99.5 ± 0.01**	92.0 ± 0.10	66.4 ± 0.35	69.7 ± 0.14
	ProGAP	∞	84.5 ± 0.24	99.3 ± 0.03	**93.3 ± 0.04**	**74.4 ± 0.14**	**73.9 ± 0.25**
Edge-Level Private	MLP	0.0	50.8 ± 0.20	82.5 ± 0.08	71.1 ± 0.18	34.9 ± 0.02	51.5 ± 0.22
	EdgeRand	1.0	50.2 ± 0.50	82.8 ± 0.05	72.7 ± 0.1	34.9 ± 0.05	52.1 ± 0.48
	GAP	1.0	69.4 ± 0.39	97.5 ± 0.06	78.8 ± 0.26	46.5 ± 0.58	62.4 ± 0.28
	ProGAP	1.0	**77.2 ± 0.33**	**97.8 ± 0.05**	**84.2 ± 0.07**	**56.9 ± 0.30**	**68.8 ± 0.23**
Node-Level Private	DP-MLP	8.0	50.2 ± 0.25	81.5 ± 0.12	73.6 ± 0.05	34.5 ± 0.13	50.8 ± 0.37
	DP-GNN	8.0	62.6 ± 0.86	**95.6 ± 0.31**	78.5 ± 0.86	46.5 ± 0.57	54.2 ± 0.73
	GAP	8.0	63.9 ± 0.49	93.9 ± 0.09	77.6 ± 0.07	43.0 ± 0.20	58.2 ± 0.39
	ProGAP	8.0	**69.3 ± 0.33**	94.0 ± 0.04	**79.1 ± 0.10**	**48.5 ± 0.36**	**61.0 ± 0.34**

FB-100, where ProGAP achieves 5.5% more accuracy points.

To examine the performance of ProGAP 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 ProGAP for each privacy budget and compare it with GAP as the most similar baseline. The outcome for both edge-level and node-level privacy settings is depicted in Figure 5.2.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 representational power than GAP, which just recursively computes the aggregations on the initial node representations.

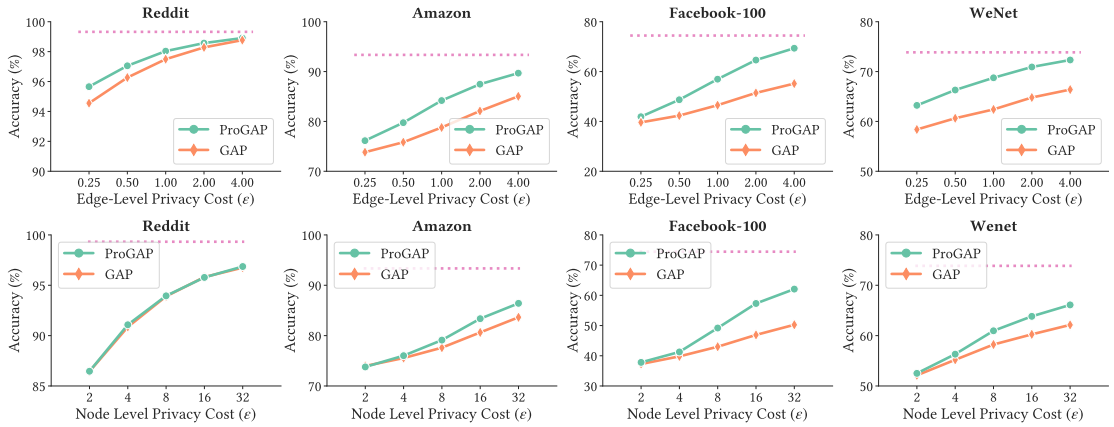


Figure 5.2: Accuracy-privacy trade-off of edge-level (top) and node-level (bottom) private methods. The dotted line represents the accuracy of the non-private ProGAP.

5.6.2 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 5.3. 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 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 (Jagielski et al., 2020; Jayaraman & Evans, 2019; Nasr et al., 2021; Sajadmanesh et al., 2023).

5.6.3 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 5.4. 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.

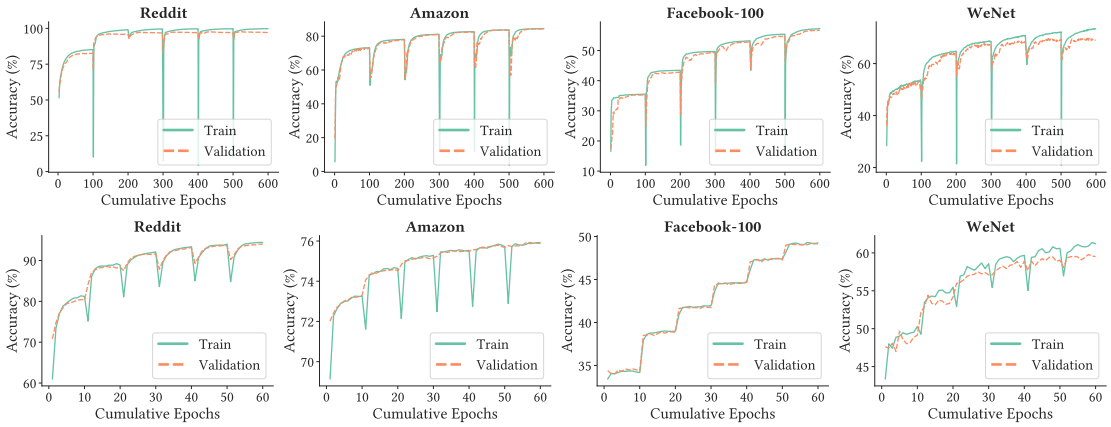


Figure 5.3: Convergence of ProGAP with $K = 5$ under edge-level (top) and node-level (bottom) privacy, with $\epsilon = 1$ and $\epsilon = 8$, respectively.

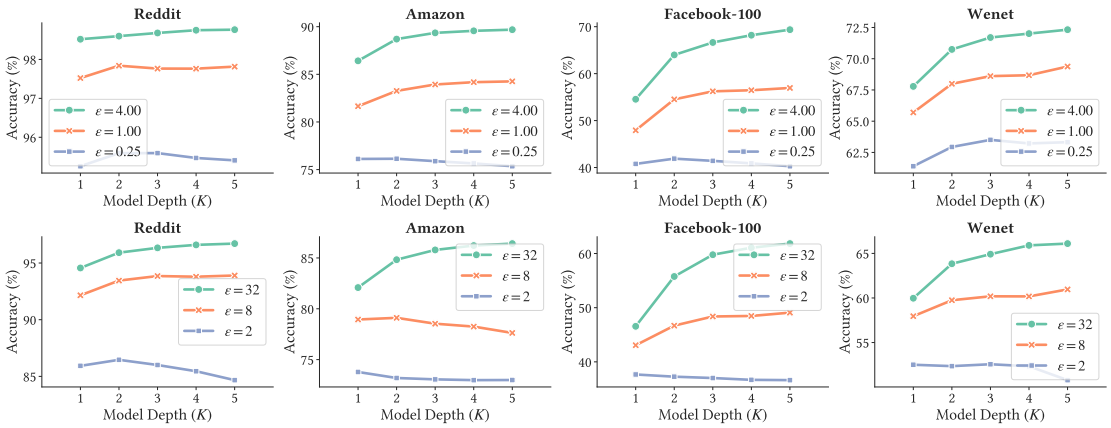


Figure 5.4: Effect of the model depth on ProGAP's accuracy under edge-level (top) and node-level (bottom) privacy.

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.

5.6.4 Progressive vs. Layerwise Training

We compare the performance of ProGAP using two different training strategies: progressive training and layerwise training. Similar to Table 5.2, 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 5.3. Overall, we observe that the progressive training yields higher accuracies than the layerwise strategy in most cases, which as mentioned in Section 5.4, is due to the higher capacity of the progressive approach.

Table 5.3: 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

5.7 Conclusion

In this chapter, 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.

6 Conclusion and Future Directions

In this chapter, we first summarize the contributions of this thesis and then discuss the impact and implications of this work. Finally, we outline some promising directions for future research in this area.

6.1 Summary of Contributions

This thesis has made significant strides in the field of privacy-preserving machine learning on graphs, with a particular emphasis on the application of differential privacy (DP) in graph neural networks (GNNs). The contributions of this thesis can be summarized as follows.

In Chapter 2, we introduced a novel privacy-preserving GNN learning framework that synergistically combines the power of Graph Convolutional Networks (GCNs) with Local Differential Privacy (LDP). This framework was designed to address the problem of node attribute privacy in GNNs, where individual nodes in the graph have potentially sensitive attributes that need to be protected. The framework introduced a mechanism to add noise to the features of each node, thereby ensuring that the sensitive attributes of the nodes are adequately protected. An unbiased mechanism was also proposed to estimate the graph convolution operation based on the noisy features which allows for effective learning of the model. The experimental results demonstrated that this framework could achieve competitive performance while providing strong privacy guarantees.

Chapter 3 delved deeper into the realm of local differential privacy with the introduction of Locally Private Graph Neural Networks (LPGNN). This innovative approach to privacy-preserving GNNs took the principles of LDP and applied them in a novel way. LPGNN introduced a unique noise injection mechanism, a new denoising graph convolutional layer, and an innovative algorithm to learn from noisy labels. The experimental results showed that LPGNN could provide robust privacy guarantees while maintaining competitive performance on a variety of graph learning tasks, demonstrating the effectiveness of this approach.

In Chapter 4, we shifted our focus to the global model of DP with the introduction of GAP, a

novel approach to differentially private GNNs. GAP brought a new perspective to the field with its aggregation perturbation technique, which injected noise directly into the aggregation process of GNNs. Combined with the proposed model decoupling approach, GAP could provide strong privacy guarantees while achieving competitive performance on a variety of graph learning tasks, highlighting the potential of differentially private GNNs in effectively learning from sensitive graph data.

Finally, Chapter 5 presented ProGAP, a progressive learning framework for GNNs that leverages the global model of DP. ProGAP built upon the work of Chapter 4 and introduced a novel progressive learning strategy. This strategy allowed for the gradual refinement of the GNN's parameters while maintaining privacy guarantees, providing a dynamic approach to learning differentially private GNNs. The experimental results showed that ProGAP could provide robust privacy guarantees while achieving state-of-the-art performance on a variety of graph learning tasks, demonstrating the potential of progressive learning strategies in privacy-preserving GNNs.

Each of these contributions represents a substantial advancement in the development of privacy-preserving techniques for machine learning on graphs, providing promising directions for future research in this field.

6.2 Impact and Implications

The research presented in this thesis has important implications for the field of privacy-preserving machine learning on graphs, which is of paramount importance as most of the today's real-world graphs, such as social networks, are generated from people's activities or represent human-related interactions. By focusing on the integration of differential privacy with graph neural networks, this work addresses a critical challenge in the field with significant implications for both academia and industry.

Firstly, this work makes a substantial contribution to the theoretical underpinnings of privacy-preserving machine learning. The innovative techniques and strategies introduced in this thesis, including the multi-bit mechanism, the aggregation perturbation technique, the KProp denoising layer, the Drop algorithm, and the progressive learning strategy, shed new light on the preservation of privacy in the context of learning from sensitive graph data. These contributions not only deepen our understanding of privacy-preserving machine learning but also offer practical tools that can be deployed across a variety of applications.

Secondly, the potential practical implications of this work are considerable. A broad spectrum of applications and services, ranging from social network and dating apps to shopping and financial platforms, that incorporate GNNs into their systems and handle privacy-sensitive graph data, stand to benefit significantly from the methods presented in this thesis. These methods provide robust privacy guarantees to users while preserving the utility of the services. For instance, by adopting locally private approaches such as LPGNN, these applications can

drastically reduce their reliance on the collection of users' personal data, opting instead for the proposed differentially private data collection mechanisms. This not only offers users provable data privacy guarantees but can also bolster the security of such services, as the volume of users' private information on the servers' data centers would be significantly reduced, thereby diminishing the incentive for potential attackers to breach their systems. By employing globally private methods, such as GAP and PROGAP, these applications can learn rich user representations from their graph data without compromising user privacy. This practice can cultivate trust between users and service providers, a critical factor in the success of any service, particularly as privacy concerns continue to grow among users.

Lastly, this research contributes to the broader initiative to foster responsible and ethical practices in machine learning, by tackling the issue of privacy in graph-based machine learning. As machine learning models become more pervasive in our daily lives and handle a growing volume of personal data, ensuring privacy evolves from being merely a technical challenge to a societal necessity. By demonstrating the feasibility of developing effective machine learning models that respect privacy, this research adds a valuable perspective to the ongoing discourse on privacy and ethics in machine learning. Therefore, the potential implications of this research extend beyond the technical realm, as they could influence policy discussions and ethical considerations related to privacy in the age of big data and artificial intelligence.

In conclusion, this thesis advances both the theory and practice of privacy-preserving machine learning on graphs. The developed techniques offer robust privacy guarantees and practical tools for a multitude of applications, fostering trust between users and service providers. As privacy concerns continue to escalate, the importance of such research intensifies. This work marks a substantial stride in the field, setting the stage for future exploration in privacy-preserving machine learning on graphs.

6.3 Future Directions

The research presented in this thesis opens up numerous avenues for future work. As we continue to grapple with the challenges of protecting privacy in an increasingly data-driven world, it is clear that the journey towards fully privacy-preserving machine learning is far from over. This section outlines several potential directions for future research in this field, each of which could further enhance the privacy and utility of machine learning on graphs.

Developing New Techniques to Enhance Utility-Privacy Trade-off. The methods developed in this thesis strike a balance between privacy and utility, but there is always room for improvement. Future research could focus on developing new practices that provide better utility for a given level of privacy, or vice versa. This could involve refining the existing approaches introduced in this thesis, such as the Drop algorithm, the aggregation perturbation technique, and the progressive learning strategy, or innovating entirely new methods that offer even stronger privacy guarantees or improved efficiency. For instance, more efficient feature or

label perturbation techniques or more sophisticated noise reduction mechanisms could be explored to improve the utility of locally private GNNs while maintaining the same level of privacy. Alternatively, new training strategies and model architectures, other than the decoupled architecture of GAP or the progressive learning paradigm of ProGAP, could be developed to enhance the accuracy-privacy trade-off of differentially private GNNs.

Exploring Different Problem Settings. The methods developed in this thesis have been applied to specific problem settings, but future work could consider a broader range of scenarios where privacy-preserving machine learning on graphs could be beneficial. This could involve different types of tasks, such as link prediction, or graph generation, or different types of data distributions, such as graphs with heterophily or graphs with community structures. Furthermore, future research could consider settings with different privacy requirements, such as settings where different nodes or edges have different levels of sensitivity, or preserving the privacy of individual communities. By exploring these different problem settings, future research could broaden the scope and applicability of the methods developed in this thesis, making privacy-preserving machine learning on graphs more versatile and effective in a wider range of scenarios.

Applying to Other Types of Graph Data. The methods developed in this thesis could be extended to other types of graph data, opening up new avenues for research in this area. For instance, dynamic graphs, where the structure and features of the graph change over time, present unique challenges for privacy preservation, especially with regard to differential privacy. Future research could explore how the methods developed in this thesis could be adapted to handle such dynamic data. Similarly, heterogeneous graphs, which contain different types of nodes and edges, could also benefit from privacy-preserving techniques. Extending the methods to handle such data could enable privacy-preserving machine learning in a wider range of applications, such as social media analysis, where different types of entities (e.g., users, posts, comments) and relationships (e.g., friendships, likes, replies) are represented in the graph. Lastly, hypergraphs, which generalize graphs by allowing edges to connect any number of nodes, could also be a fruitful area for future research. Privacy-preserving techniques for hypergraphs could be useful in applications such as co-authorship networks and protein interaction networks, where relationships between more than two entities are common. Exploring these different types of graph data could lead to the development of more effective privacy-preserving techniques for machine learning on graphs.

Studying the Interplay Between Privacy and Other Notions of Trustworthiness. Future research could also delve into the relationship between privacy and other crucial aspects of trustworthiness in machine learning, such as robustness, fairness, and explainability. For instance, it would be interesting to investigate how the incorporation of privacy-preserving techniques influences the robustness of a model against adversarial attacks. Is it possible to

construct a model that maintains robustness while ensuring privacy? Similarly, the impact of privacy preservation on the fairness of a model is another intriguing area of study. Could we design a model that upholds fairness principles while also safeguarding privacy, ensuring that it does not discriminate against certain groups? Lastly, the effect of privacy preservation on the explainability of a model is a compelling topic. Is there a pathway to create a model that is both private and capable of providing users with understandable explanations for its predictions? Unraveling how these different aspects interact and how to balance them could lead to the development of more trustworthy and effective machine learning systems.

Bibliography

Abadal, S., Jain, A., Guirado, R., López-Alonso, J., & Alarcón, E. (2021). Computing graph neural networks: A survey from algorithms to accelerators. *ACM Computing Surveys (CSUR)*, *54*(9), 1–38.

Abadi, M. í. n., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., & Zhang, L. (2016). Deep learning with differential privacy. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 308–318.

Abu-El-Haija, S., Perozzi, B., Kapoor, A., Alipourfard, N., Lerman, K., Harutyunyan, H., Steeg, G. V., & Galstyan, A. (2019, June). M ix H op: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In K. Chaudhuri & R. Salakhutdinov (Eds.). PMLR.

Acharya, J., Sun, Z., & Zhang, H. (2018). Communication efficient, sample optimal, linear time locally private discrete distribution estimation. *arXiv preprint arXiv:1802.04705*.

Acharya, J., Sun, Z., & Zhang, H. (2019). Hadamard response: Estimating distributions privately, efficiently, and with little communication. *The 22nd International Conference on Artificial Intelligence and Statistics*, 1120–1129.

Agrawal, N., Shahin Shamsabadi, A., Kusner, M. J., & Gasc ó n, A. à. (2019). Quotient: Two-party secure neural network training and prediction. *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 1231–1247.

Ahmedt-Aristizabal, D., Armin, M. A., Denman, S., Fookes, C., & Petersson, L. (2021). Graph-based deep learning for medical diagnosis and analysis: Past, present and future. *arXiv preprint arXiv:2105.13137*.

Ayle, M., Schuchardt, J., Gosch, L., Zügner, D., & Günnemann, S. (2023). Training differentially private graph neural networks with random walk sampling. *arXiv preprint arXiv:2301.00738*.

Balle, B., & Wang, Y.-X. (2018). Improving the gaussian mechanism for differential privacy: Analytical calibration and optimal denoising. *International Conference on Machine Learning*, 394–403.

Bang, W., Xiangwen, Y., Shirui, P., & Xingliang, Y. (2021). Adapting membership inference attacks to gnn for graph classification: Approaches and implications. *2021 IEEE International Conference on Data Mining (ICDM)*.

Bassily, R., Nissim, K., Stemmer, U., & Thakurta, A. (2017). Practical locally private heavy hitters. *arXiv preprint arXiv:1707.04982*.

Bassily, R., & Smith, A. (2015). Local, private, efficient protocols for succinct histograms. *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, 127–135.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. (2018). Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*.

Belilovsky, E., Eickenberg, M., & Oyallon, E. (2020, July). Decoupled greedy learning of CNNs. In H. D. III & A. Singh (Eds.), *Proceedings of the 37th international conference on machine learning* (pp. 736–745, Vol. 119). PMLR.

Bishop, C. M., & Nasrabadi, N. M. (2006). *Pattern recognition and machine learning* (Vol. 4). Springer.

Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., Ramage, D., Segal, A., & Seth, K. (2017). Practical secure aggregation for privacy-preserving machine learning. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 1175–1191.

Bun, M., Nelson, J., & Stemmer, U. (2019). Heavy hitters and the structure of local privacy. *ACM Transactions on Algorithms (TALG)*, *15*(4), 1–40.

Carlini, N., Liu, C., Erlingsson, Ú., Kos, J., & Song, D. (2019). The secret sharer: Evaluating and testing unintended memorization in neural networks. *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 267–284.

Chami, I., Abu-El-Haija, S., Perozzi, B., Ré, C., & Murphy, K. (2022). Machine learning on graphs: A model and comprehensive taxonomy. *The Journal of Machine Learning Research*, *23*(1), 3840–3903.

Chen, C., Hu, W., Xu, Z., & Zheng, Z. (2021). Fedgl: Federated graph learning framework with global self-supervision. *arXiv preprint arXiv:2105.03170*.

Chen, F., Li, P., Miyazaki, T., & Wu, C. (2021). Fedgraph: Federated graph learning with intelligent sampling. *IEEE Transactions on Parallel and Distributed Systems*.

Chen, Z., Li, X., & Bruna, J. (2017). Supervised community detection with line graph neural networks. *arXiv preprint arXiv:1705.08415*.

Cheung, M., & Moura, J. M. F. (2020). Graph neural networks for covid-19 drug discovery. *2020 IEEE International Conference on Big Data (Big Data)*, 5646–5648. https://doi.org/10.1109/BigData50022.2020.9378164

Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S., & Hsieh, C.-J. (2019). Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 257–266.

Cormode, G., Kulkarni, T., & Srivastava, D. (2018). Marginal release under local differential privacy. *Proceedings of the 2018 International Conference on Management of Data*, 131–146.

Corso, G., Cavalleri, L., Beaini, D., Liò, P., & Veličković, P. (2020). Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*.

Daigavane, A., Madan, G., Sinha, A., Thakurta, A. G., Aggarwal, G., & Jain, P. (2021). Node-level differentially private graph neural networks. *arXiv preprint arXiv:2111.15521*.

Diao, Z., Wang, X., Zhang, D., Liu, Y., Xie, K., & He, S. (2019). Dynamic spatial-temporal graph convolutional neural networks for traffic forecasting. *Proceedings of the AAAI conference on artificial intelligence*, *33*, 890–897.

Differential Privacy Team, A. (n.d.). Learning with privacy at scale [https://apple.co/3di4jjP].

Ding, B., Kulkarni, J., & Yekhanin, S. (2017). Collecting telemetry data privately. *Advances in Neural Information Processing Systems*, 3571–3580.

Duchi, J. C., Jordan, M. I., & Wainwright, M. J. (2018). Minimax optimal procedures for locally private estimation. *Journal of the American Statistical Association*, *113*(521), 182–201.

Duddu, V., Boutet, A., & Shejwalkar, V. (2020). Quantifying privacy leakage in graph embedding. *MobiQuitous 2020 - 17th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, 76–85. https://doi.org/10.1145/3448891.3448939

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A. á. n., & Adams, R. P. (2015). Convolutional networks on graphs for learning molecular fingerprints. *Advances in neural information processing systems*, 2224–2232.

Dwork, C. (2008). Differential privacy: A survey of results. *International conference on theory and applications of models of computation*, 1–19.

Dwork, C., McSherry, F., Nissim, K., & Smith, A. (2006). Calibrating noise to sensitivity in private data analysis. *Theory of cryptography conference*, 265–284.

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

Erlingsson, Ú. l., Pihur, V., & Korolova, A. (2014). Rappor: Randomized aggregatable privacy-preserving ordinal response. *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, 1054–1067.

Evans, D., Kolesnikov, V., Rosulek, M., et al. (2018). A pragmatic introduction to secure multi-party computation. *Foundations and Trends ® in Privacy and Security*, *2*(2-3), 70–246.

Falcon, W. (2019). Pytorch lightning. *GitHub. Note: https://github. com/williamFalcon/pytorch-lightning Cited by*, *3*.

Fey, M., & Lenssen, J. E. (2019). Fast graph representation learning with PyTorch Geometric. *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Fey, M., Lenssen, J. E., Weichert, F., & Leskovec, J. (2021). Gnnautoscale: Scalable and expressive graph neural networks via historical embeddings. *International Conference on Machine Learning*, 3294–3304.

Fortunato, S. (2010). Community detection in graphs. *Physics reports*, *486*(3-5), 75–174.

Frasca, F., Rossi, E., Eynard, D., Chamberlain, B., Bronstein, M., & Monti, F. (2020). Sign: Scalable inception graph neural networks. *ICML 2020 Workshop on Graph Representation Learning and Beyond*.

Fredrikson, M., Jha, S., & Ristenpart, T. (2015). Model inversion attacks that exploit confidence information and basic countermeasures. *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 1322–1333.

Gaboardi, M., & Rogers, R. (2018). Local private hypothesis testing: Chi-square tests. *International Conference on Machine Learning*, 1626–1635.

Giunchiglia, F., Bison, I., Busso, M., Chenu-Abente, R., Rodas, M., Zeni, M., Gunel, C., Veltri, G., De Götzen, A., Kun, P., et al. (2021). A worldwide diversity pilot on daily routines and social practices (2020).

Gkalelis, N., Goulas, A., Galanopoulos, D., & Mezaris, V. (2021). Objectgraphs: Using objects and a graph convolutional network for the bottom-up recognition and explanation of events in video. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 3375–3383.

Gupta, O., & Raskar, R. (2018). Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications*, *116*, 1–8.

Gylberth, R., Adnan, R., Yazid, S., & Basaruddin, T. (2017). Differentially private optimization algorithms for deep neural networks. *2017 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, 387–394.

Hamilton, W. L., Ying, R., & Leskovec, J. (2017a). Inductive representation learning on large graphs. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 1025–1035.

Hamilton, W. L., Ying, R., & Leskovec, J. (2017b). Representation learning on graphs: Methods and applications. *IEEE Data Engineering Bulletin*.

Hay, M., Li, C., Miklau, G., & Jensen, D. (2009). Accurate estimation of the degree distribution of private networks. *2009 Ninth IEEE International Conference on Data Mining*, 169–178.

He, C., Balasubramanian, K., Ceyani, E., Yang, C., Xie, H., Sun, L., He, L., Yang, L., Yu, P. S., Rong, Y., et al. (2021). Fedgraphnn: A federated learning system and benchmark for graph neural networks. *arXiv preprint arXiv:2104.07145*.

He, C., Li, S., Soltanolkotabi, M., & Avestimehr, S. (2021). Pipetransformer: Automated elastic pipelining for distributed training of transformers. *arXiv preprint arXiv:2102.03161*.

He, X., Jia, J., Backes, M., Gong, N. Z., & Zhang, Y. (2021a). Stealing links from graph neural networks. *30th {USENIX} Security Symposium ({USENIX} Security 21)*.

He, X., Jia, J., Backes, M., Gong, N. Z., & Zhang, Y. (2021b). Stealing Links from Graph Neural Networks. *USENIX Security Symposium (USENIX Security)*.

He, X., Wen, R., Wu, Y., Backes, M., Shen, Y., & Zhang, Y. (2021). Node-level membership inference attacks against graph neural networks. *arXiv preprint arXiv:2102.05429*.

Hesamifard, E., Takabi, H., & Ghasemi, M. (2017). Cryptodl: Deep neural networks over encrypted data. *arXiv preprint arXiv:1711.05189*.

Hesamifard, E., Takabi, H., Ghasemi, M., & Wright, R. N. (2018). Privacy-preserving machine learning as a service. *Proceedings on Privacy Enhancing Technologies*, *2018*(3), 123–142.

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association, 58*(301), 13–30. https://doi.org/10.1080/01621459.1963.10500830

Hsieh, I.-C., & Li, C.-T. (2021). Netfense: Adversarial defenses against privacy attacks on neural networks for graph data. *IEEE Transactions on Knowledge and Data Engineering*.

Hsu, J., Gaboardi, M., Haeberlen, A., Khanna, S., Narayan, A., Pierce, B. C., & Roth, A. (2014). Differential privacy: An economic method for choosing epsilon. *2014 IEEE 27th Computer Security Foundations Symposium*, 398–410.

Imola, J., Murakami, T., & Chaudhuri, K. (2022). Communication-Efficient triangle counting under local differential privacy. *31st USENIX Security Symposium (USENIX Security 22)*.

Jagielski, M., Ullman, J. R., & Oprea, A. (2020). Auditing differentially private machine learning: How private is private sgd? *Proceedings of the Advances in Neural Information Processing (NeurIPS)*.

Jayaraman, B., & Evans, D. (2019). Evaluating differentially private machine learning in practice. *28th USENIX Security Symposium (USENIX Security 19)*, 1895–1912.

Jha, M., & Raskhodnikova, S. (2013). Testing and reconstruction of lipschitz functions with applications to data privacy. *SIAM Journal on Computing, 42*(2), 700–731.

Jiang, M., Jung, T., Karl, R., & Zhao, T. (2020). Federated dynamic gnn with secure aggregation. *arXiv preprint arXiv:2009.07351*.

Jiang, W., & Luo, J. (2022). Graph neural network for traffic forecasting: A survey. *Expert Systems with Applications*, 117921.

Kairouz, P., Bonawitz, K., & Ramage, D. (2016). Discrete distribution estimation under local privacy. *International Conference on Machine Learning*, 2436–2444.

Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., et al. (2019). Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*.

Karras, T., Aila, T., Laine, S., & Lehtinen, J. (2017). Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*.

Kasiviswanathan, S. P., Lee, H. K., Nissim, K., Raskhodnikova, S., & Smith, A. (2011). What can we learn privately? *SIAM Journal on Computing, 40*(3), 793–826.

Kearnes, S., McCloskey, K., Berndl, M., Pande, V., & Riley, P. (2016). Molecular graph convolutions: Moving beyond fingerprints. *Journal of computer-aided molecular design, 30*(8), 595–608.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kipf, T. N., & Welling, M. (2016). Variational graph auto-encoders. *NIPS Workshop on Bayesian Deep Learning*.

Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations (ICLR)*.

Klambauer, G., Unterthiner, T., Mayr, A., & Hochreiter, S. (2017). Self-normalizing neural networks. *Proceedings of the 31st international conference on neural information processing systems*, 972–981.

Klicpera, J., Wei ß enberger, S., & G ü nnemann, S. (2019). Diffusion improves graph learning. *Advances in Neural Information Processing Systems*, 13354–13366.

Kolluri, A., Baluta, T., Hooi, B., & Saxena, P. (2022). Lpgnet: Link private graph networks for node classification. *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 1813–1827. https://doi.org/10.1145/3548606.3560705

Konečný, J., McMahan, H. B., Yu, F. X., Richtarik, P., Suresh, A. T., & Bacon, D. (2016). Federated learning: Strategies for improving communication efficiency. *NIPS Workshop on Private Multi-Party Machine Learning*.

Lee, J., & Clifton, C. (2011). How much is enough? choosing ϵ for differential privacy. In X. Lai, J. Zhou, & H. Li (Eds.), *Information security* (pp. 325–340). Springer Berlin Heidelberg.

Li, K., Luo, G., Ye, Y., Li, W., Ji, S., & Cai, Z. (2020). Adversarial privacy-preserving graph embedding against inference attack. *IEEE Internet of Things Journal*, *8*(8), 6904–6915.

Li, Q., Han, Z., & Wu, X.-M. (2018). Deeper insights into graph convolutional networks for semi-supervised learning. *arXiv preprint arXiv:1801.07606*.

Li, T., Sahu, A. K., Talwalkar, A., & Smith, V. (2019). Federated learning: Challenges, methods, and future directions. *arXiv preprint arXiv:1908.07873*.

Li, Y., Chen, L., et al. (2021). Unified robust training for graph neuralnetworks against label noise. *arXiv preprint arXiv:2103.03414*.

Li, Y., Zemel, R., Brockschmidt, M., & Tarlow, D. (2016). Gated graph sequence neural networks. *Proceedings of ICLR'16*.

Li, Z., Murkute, J. V., Gyawali, P. K., & Wang, L. (2020). Progressive learning and disentanglement of hierarchical representations. *arXiv preprint arXiv:2002.10549*.

Liao, P., Zhao, H., Xu, K., Jaakkola, T., Gordon, G., Jegelka, S., & Salakhutdinov, R. (2020). Graph adversarial networks: Protecting information against adversarial attacks. *arXiv preprint arXiv:2009.13504*.

Liao, P., Zhao, H., Xu, K., Jaakkola, T., Gordon, G. J., Jegelka, S., & Salakhutdinov, R. (2021). Information obfuscation of graph neural networks. *International Conference on Machine Learning*, 6600–6610.

Liben-Nowell, D., & Kleinberg, J. (2003). The link prediction problem for social networks. *Proceedings of the twelfth international conference on Information and knowledge management*, 556–559.

Lin, W., Li, B., & Wang, C. (2022). Towards private learning on decentralized graphs with local differential privacy. *IEEE Transactions on Information Forensics and Security*, *17*, 2936–2946. https://doi.org/10.1109/TIFS.2022.3198283

Liu, B., Ding, M., Shaham, S., Rahayu, W., Farokhi, F., & Lin, Z. (2020). When machine learning meets privacy: A survey and outlook. *arXiv preprint arXiv:2011.11819*.

Liu, Z., Yang, L., Fan, Z., Peng, H., & Yu, P. S. (2021). Federated social recommendation with graph neural network. *arXiv preprint arXiv:2111.10778*.

McPherson, M., Smith-Lovin, L., & Cook, J. M. (2001). Birds of a feather: Homophily in social networks. *Annual review of sociology*, *27*(1), 415–444.

Meegahapola, L., Droz, W., Kun, P., de Götzen, A., Nutakki, C., Diwakar, S., Correa, S. R., Song, D., Xu, H., Bidoglia, M., et al. (2023). 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.

Mei, G., Guo, Z., Liu, S., & Pan, L. (2019). Sgnn: A graph neural network based federated learning approach by hiding structure. *2019 IEEE International Conference on Big Data (Big Data)*, 2560–2568.

Meng, C., Rambhatla, S., & Liu, Y. (2021). Cross-node federated graph neural network for spatio-temporal data modeling. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 1202–1211. https://doi.org/10.1145/3447548.3467371

Mironov, I. (2017). Rényi differential privacy. *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, 263–275.

Mirshghallah, F., Taram, M., Vepakomma, P., Singh, A., Raskar, R., & Esmaeilzadeh, H. (2020). Privacy in deep learning: A survey. *arXiv preprint arXiv:2004.12254*.

Mohammadshahi, A., & Henderson, J. (2020). Graph-to-graph transformer for transition-based dependency parsing. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, 3278–3289.

Mohassel, P., & Zhang, Y. (2017). Secureml: A system for scalable privacy-preserving machine learning. *IACR Cryptology ePrint Archive*, *2017*, 396.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., & Grohe, M. (2019). Weisfeiler and leman go neural: Higher-order graph neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*, 4602–4609.

Mueller, T. T., Paetzold, J. C., Prabhakar, C., Usynin, D., Rueckert, D., & Kaissis, G. (2022). Differentially private graph neural networks for whole-graph classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Narayanan, A., & Shmatikov, V. (2008). Robust de-anonymization of large sparse datasets. *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, 111–125.

Nasr, M., Song, S., Thakurta, A., Papernot, N., & Carlini, N. (2021). Adversary instantiation: Lower bounds for differentially private machine learning. *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*.

Newman, M. (2018). *Networks*. Oxford university press.

Ni, X., Xu, X., Lyu, L., Meng, C., & Wang, W. (2021). A vertical federated learning framework for graph convolutional network. *arXiv preprint arXiv:2106.11593*.

Nissim, K., & Stemmer, U. (2018). Clustering algorithms for the centralized and local models. *Algorithmic Learning Theory*, 619–653.

NT, H., Jin, C. J., & Murata, T. (2019). Learning graph neural networks with noisy labels. *arXiv preprint arXiv:1905.01591*.

Olatunji, I. E., Funke, T., & Khosla, M. (2021). Releasing graph neural networks with differential privacy guarantees. *arXiv preprint arXiv:2109.08907*.

Olatunji, I. E., Nejdl, W., & Khosla, M. (2021). Membership inference attack on graph neural networks. *2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, 11–20.

Papernot, N., Abadi, M., Erlingsson, U., Goodfellow, I., & Talwar, K. (2016). Semi-supervised knowledge transfer for deep learning from private training data. *arXiv preprint arXiv:1610.05755*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc.

Patrini, G., Rozza, A., Krishna Menon, A., Nock, R., & Qu, L. (2017). Making deep neural networks robust to label noise: A loss correction approach. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1944–1952.

Paverd, A., Martin, A., & Brown, I. (2014). Modelling and automatically analysing privacy properties for honest-but-curious adversaries. *Tech. Rep.*

Pei, Y., Mao, R., Liu, Y., Chen, C., Xu, S., Qiang, F., & Tech, B. E. (2021). Decentralized federated graph neural networks. *International Workshop on Federated and Transfer Learning for Data Sparsity and Confidentiality in Conjunction with IJCAI*.

Qin, Z., Yang, Y., Yu, T., Khalil, I., Xiao, X., & Ren, K. (2016). Heavy hitter estimation over set-valued data with local differential privacy. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 192–203.

Qiu, J., Tang, J., Ma, H., Dong, Y., Wang, K., & Tang, J. (2018). Deepinf: Social influence prediction with deep learning. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2110–2119.

Raskhodnikova, S., & Smith, A. (2016). Differentially private analysis of graphs. *Encyclopedia of Algorithms*.

Rhee, S., Seo, S., & Kim, S. (2017). Hybrid approach of relation network and localized graph convolutional filtering for breast cancer subtype classification. *arXiv preprint arXiv:1711.05859*.

Rouhani, B. D., Riazi, M. S., & Koushanfar, F. (2017). Deepsecure: Scalable provably-secure deep learning. *arXiv preprint arXiv:1705.08963*.

Rozemberczki, B., Allen, C., & Sarkar, R. (2019). Multi-scale attributed node embedding. *arXiv preprint arXiv:1909.13021*.

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

Sajadmanesh, S., & Gatica-Perez, D. (2020). When differential privacy meets graph neural networks. *arXiv preprint arXiv:2006.05535v3*.

Sajadmanesh, S., & Gatica-Perez, D. (2021). Locally private graph neural networks. *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2130–2145.

Sajadmanesh, S., & Gatica-Perez, D. (2023). Progap: Progressive graph neural networks with differential privacy guarantees. *arXiv preprint arXiv:2304.08928*.

Sajadmanesh, S., Shamsabadi, A. S., Bellet, A., & Gatica-Perez, D. (2023). Gap: Differentially private graph neural networks with aggregation perturbation. *32nd USENIX Security Symposium (USENIX Security 23)*.

Sathya, S. S., Vepakomma, P., Raskar, R., Ramachandra, R., & Bhattacharya, S. (2018). A review of homomorphic encryption libraries for secure computation. *arXiv preprint arXiv:1812.02428*.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2008). The graph neural network model. *IEEE Transactions on Neural Networks*, *20*(1), 61–80.

Shamir, A. (1979). How to share a secret. *Communications of the ACM*, *22*(11), 612–613.

Sheikh, R., & Mishra, D. K. (2010). Protocols for getting maximum value for multi-party computations. *2010 Fourth Asia International Conference on Mathematical/Analytical Modelling and Computer Simulation*, 597–600.

Shokri, R., Stronati, M., Song, C., & Shmatikov, V. (2017). Membership inference attacks against machine learning models. *2017 IEEE Symposium on Security and Privacy (SP)*, 3–18.

Shringarpure, S. S., & Bustamante, C. D. (2015). Privacy risks from genomic data-sharing beacons. *The American Journal of Human Genetics*, *97*(5), 631–646.

Song, H., Kim, M., Park, D., & Lee, J.-G. (2020). Learning from noisy labels with deep neural networks: A survey. *arXiv preprint arXiv:2007.08199*.

Song, S., Chaudhuri, K., & Sarwate, A. D. (2013). Stochastic gradient descent with differentially private updates. *2013 IEEE Global Conference on Signal and Information Processing*, 245–248.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1–9.

Tang, J., Korolova, A., Bai, X., Wang, X., & Wang, X. (2017). Privacy loss in apple's implementation of differential privacy on macos 10.12. *arXiv preprint arXiv:1709.02753*.

Thakurta, A. G., Vyrros, A. H., Vaishampayan, U. S., Kapoor, G., Freudiger, J., Sridhar, V. R., & Davidson, D. (2017, March). Learning new words [US Patent 9,594,741].

Traud, A. L., Mucha, P. J., & Porter, M. A. (2012). Social structure of facebook networks. *Physica A: Statistical Mechanics and its Applications*, *391*(16), 4165–4180.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph Attention Networks. *International Conference on Learning Representations*.

Vepakomma, P., Swedish, T., Raskar, R., Gupta, O., & Dubey, A. (2018). No peek: A survey of private distributed deep learning. *arXiv preprint arXiv:1812.03288*.

Wang, B., Guo, J., Li, A., Chen, Y., & Li, H. (2021). Privacy-preserving representation learning on graphs: A mutual information perspective. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 1667–1676.

Wang, B., Li, A., Li, H., & Chen, Y. (2020). Graphfl: A federated learning framework for semi-supervised node classification on graphs. *arXiv preprint arXiv:2012.04187*.

Wang, H., & Leskovec, J. (2020). Unifying graph convolutional neural networks and label propagation. *arXiv preprint arXiv:2002.06755*.

Wang, H.-P., Stich, S., He, Y., & Fritz, M. (2022). Progfed: Effective, communication, and computation efficient federated learning by progressive training. *International Conference on Machine Learning*, 23034–23054.

Wang, J., Zhang, S., Xiao, Y., & Song, R. (2021). A review on graph neural network methods in financial applications. *arXiv preprint arXiv:2111.15367*.

Wang, N., Xiao, X., Yang, Y., Hoang, T. D., Shin, H., Shin, J., & Yu, G. (2018). Privtrie: Effective frequent term discovery under local differential privacy. *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, 821–832.

Wang, N., Xiao, X., Yang, Y., Zhao, J., Hui, S. C., Shin, H., Shin, J., & Yu, G. (2019). Collecting and analyzing multidimensional data with local differential privacy. *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, 638–649.

Wang, S., Huang, L., Wang, P., Nie, Y., Xu, H., Yang, W., Li, X.-Y., & Qiao, C. (2016). Mutual information optimally local private discrete distribution estimation. *arXiv preprint arXiv:1607.08025*.

Wang, T., Blocki, J., Li, N., & Jha, S. (2017). Locally differentially private protocols for frequency estimation. *26th {USENIX} Security Symposium ({USENIX} Security 17)*, 729–745.

Wang, T., Li, N., & Jha, S. (2018). Locally differentially private frequent itemset mining. *2018 IEEE Symposium on Security and Privacy (SP)*, 127–143.

Wang, T., Li, N., & Jha, S. (2019). Locally differentially private heavy hitter identification. *IEEE Transactions on Dependable and Secure Computing*.

Wang, Y., Perazzi, F., McWilliams, B., Sorkine-Hornung, A., Sorkine-Hornung, O., & Schroers, C. (2018). A fully progressive approach to single-image super-resolution. *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 864–873.

Wang, Y., Wu, X., & Hu, D. (2016). Using randomized response for differential privacy preserving data collection. *EDBT/ICDT Workshops*, *1558*, 0090–6778.

Wang, Y.-X., Balle, B., & Kasiviswanathan, S. P. (2019, April). Subsampled renyi differential privacy and analytical moments accountant. In K. Chaudhuri & M. Sugiyama (Eds.), *Proceedings of the twenty-second international conference on artificial intelligence and statistics* (pp. 1226–1235, Vol. 89). PMLR.

Wu, B., Yang, X., Pan, S., & Yuan, X. (2020). Model extraction attacks on graph neural networks: Taxonomy and realization. *arXiv preprint arXiv:2010.12751*.

Wu, C., Wu, F., Cao, Y., Huang, Y., & Xie, X. (2021). Fedgnn: Federated graph neural network for privacy-preserving recommendation. *arXiv preprint arXiv:2102.04925*.

Wu, F., Long, Y., Zhang, C., & Li, B. (2021). Linkteller: Recovering private edges from graph neural networks via influence analysis. *arXiv preprint arXiv:2108.06504*.

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., & Weinberger, K. (2019). Simplifying graph convolutional networks. *International conference on machine learning*, 6861–6871.

Wu, R., Zhang, G., Lu, S., & Chen, T. (2020). Cascade ef-gan: Progressive facial expression editing with local focuses. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 5021–5030.

Wu, X., Li, F., Kumar, A., Chaudhuri, K., Jha, S., & Naughton, J. (2017). Bolt-on differential privacy for scalable stochastic gradient descent-based analytics. *Proceedings of the 2017 ACM International Conference on Management of Data*, 1307–1322.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Philip, S. Y. (2020). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*.

Xie, H., Ma, J., Xiong, L., & Yang, C. (2021). Federated graph classification over non-iid graphs. *Advances in Neural Information Processing Systems*, *34*.

Xu, D., Yuan, S., Wu, X., & Phan, H. (2018). Dpne: Differentially private network embedding. *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 235–246.

Xu, K., Hu, W., Leskovec, J., & Jegelka, S. (2019). How powerful are graph neural networks? *International Conference on Learning Representations*.

Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.-i., & Jegelka, S. (2018, July). Representation learning on graphs with jumping knowledge networks. In J. Dy & A. Krause (Eds.), *Proceedings of the 35th international conference on machine learning* (pp. 5453–5462, Vol. 80). PMLR.

Yang, Z., Cohen, W. W., & Salakhutdinov, R. (2016). Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861*.

Ye, M., & Barg, A. (2018). Optimal schemes for discrete distribution estimation under locally differential privacy. *IEEE Transactions on Information Theory*, *64*(8), 5662–5676.

Yi, K., & Wu, J. (2019). Probabilistic end-to-end noise correction for learning with noisy labels. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 7017–7025.

Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., & Leskovec, J. (2018). Graph convolutional neural networks for web-scale recommender systems. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 974–983.

You, J., Ying, Z., & Leskovec, J. (2020). Design space for graph neural networks. *Advances in Neural Information Processing Systems*, *33*.

Yousefpour, A., Shilov, I., Sablayrolles, A., Testuggine, D., Prasad, K., Malek, M., Nguyen, J., Ghosh, S., Bharadwaj, A., Zhao, J., Cormode, G., & Mironov, I. (2021). Opacus: User-friendly differential privacy library in PyTorch. *arXiv preprint arXiv:2109.12298*.

Zeng, H., Zhou, H., Srivastava, A., Kannan, R., & Prasanna, V. (2019). Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*.

Zhang, H., Cisse, M., Dauphin, Y. N., & Lopez-Paz, D. (2017). Mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*.

Zhang, H., Shen, T., Wu, F., Yin, M., Yang, H., & Wu, C. (2021). Federated graph learning–a position paper. *arXiv preprint arXiv:2105.11099*.

Zhang, K., Yang, C., Li, X., Sun, L., & Yiu, S. M. (2021). Subgraph federated learning with missing neighbor generation. *Advances in Neural Information Processing Systems*, *34*.

Zhang, M., & Chen, Y. (2018). Link prediction based on graph neural networks. *Advances in Neural Information Processing Systems*, 5165–5175.

Zhang, M., Li, P., Xia, Y., Wang, K., & Jin, L. (2021). Labeling trick: A theory of using graph neural networks for multi-node representation learning. *Advances in Neural Information Processing Systems*, *34*.

Zhang, S., & Ni, W. (2019). Graph embedding matrix sharing with differential privacy. *IEEE Access*, *7*, 89390–89399.

Zhang, Y., Pal, S., Coates, M., & Ustebay, D. (2019). Bayesian graph convolutional neural networks for semi-supervised classification. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*, 5829–5836.

Zhang, Z., Cui, P., & Zhu, W. (2022). Deep learning on graphs: A survey. *IEEE Transactions on Knowledge & Data Engineering*, *34*(01), 249–270. https://doi.org/10.1109/TKDE.2020.2981333

Zhang, Z., Liu, Q., Huang, Z., Wang, H., Lee, C.-K., & Chen, E. (2022). Model inversion attacks against graph neural networks. *IEEE Transactions on Knowledge and Data Engineering*.

Zhang, Z., Liu, Q., Huang, Z., Wang, H., Lu, C., Liu, C., & Chen, E. (2021a, August). Graphmi: Extracting private graph data from graph neural networks [Main Track]. In Z.-H. Zhou (Ed.), *Proceedings of the thirtieth international joint conference on artificial intelligence, IJCAI-21* (pp. 3749–3755). International Joint Conferences on Artificial Intelligence Organization. https://doi.org/10.24963/ijcai.2021/516

Zhang, Z., Liu, Q., Huang, Z., Wang, H., Lu, C., Liu, C., & Chen, E. (2021b, August). Graphmi: Extracting private graph data from graph neural networks. In Z.-H. Zhou (Ed.), *Proceedings of the thirtieth international joint conference on artificial intelligence, IJCAI-21* (pp. 3749–3755). International Joint Conferences on Artificial Intelligence Organization. https://doi.org/10.24963/ijcai.2021/516

Zhang, Z., Chen, M., Backes, M., Shen, Y., & Zhang, Y. (2022). Inference attacks against graph neural networks. *31st USENIX Security Symposium (USENIX Security 22)*.

Zhang, Z., & Sabuncu, M. R. (2018). Generalized cross entropy loss for training deep neural networks with noisy labels. *arXiv preprint arXiv:1805.07836*.

Zhao, L., Wang, Q., Zou, Q., Zhang, Y., & Chen, Y. (2019). Privacy-preserving collaborative deep learning with unreliable participants. *IEEE Transactions on Information Forensics and Security*, *15*, 1486–1500.

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., & Sun, M. (2020). Graph neural networks: A review of methods and applications. *AI Open*, *1*, 57–81.

Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., & Sun, M. (2018). Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*.

Zhou, J., Chen, C., Zheng, L., Zheng, X., Wu, B., Liu, Z., & Wang, L. (2020). Privacy-preserving graph neural network for node classification. *arXiv preprint arXiv:2005.11903*.

Zhu, T., Ye, D., Wang, W., Zhou, W., & Yu, P. (2020). More than privacy: Applying differential privacy in key areas of artificial intelligence. *IEEE Transactions on Knowledge & Data Engineering*, *1*(01), 1–1.

Zhu, Y., & Wang, Y.-X. (2019, June). Poission subsampled rényi differential privacy. In K. Chaudhuri & R. Salakhutdinov (Eds.), *Proceedings of the 36th international conference on machine learning* (pp. 7634–7642, Vol. 97). PMLR.

Sina Sajadmanesh

linkedin.com/in/sajadmanesh

sina.sajadmanesh@gmail.com

(+41) 787812478

Lausanne, Switzerland

Summary

Machine Learning Researcher – Software Engineer – Data Scientist

- **A self-motivated and diversely skilled professional** with 8 years of experience in designing, developing, and implementing machine learning models, AI solutions, and data-driven insights.
- **Adept in various machine learning frameworks, programming languages, and data analysis tools.** Passionate about leveraging cutting-edge technology to solve complex real-world problems and drive business growth.
- **A proven track record of success in research and development**, with several publications in top-tier conferences and journals. Demonstrated team working and leading projects from concept to completion.
- **A continuous learner with a growth mindset**, eager to contribute to the development of next-generation AI solutions and software that align with the strategic objectives and vision of a leading technology organization.

Skills and Expertise

Programming and Scripting Languages:
Python (Expert), Java (Moderate), C++ (Moderate), Shell Scripting (Moderate), SQL (Moderate), LaTeX (Moderate)

Machine Learning & Data Analysis:
PyTorch, PyTorch-Geometric, Tensorflow, Tensorflow-Lite, Scikit-Learn, Pandas, Numpy, Jupyter, Matplotlib

MLOps & Related Technologies:
Weights & Biases, PyTorch-Lightning, Dask, Git, Linux, Docker

Privacy-Enhancing Technologies:
Flower (Federated Learning), Opacus (Differentially Private ML), Auto-DP (Privacy Accounting)

Education

Ph.D. in Electrical Engineering (GPA: 5.75 / 6)
Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland — May 2019 – Aug 2023

M.Sc. in Information Technology Engineering (GPA: 18.1 / 20)
Sharif University of Technology, Tehran, Iran — Sep 2014 – Sep 2016

B.Sc. in Computer Software Engineering (GPA: 16.19 / 20)
University of Isfahan, Esfahan, Iran — Sep 2009 – Feb 2014

Professional Experience

Research Assistant
Idiap Research Institute, Martigny, Switzerland, — May 2019 – August 2023

Recruited as a doctoral researcher to contribute to AI4Media, a European research project focusing on ethical and trustworthy AI to serve media and society. Developed privacy-preserving graph neural network (GNN) models using differential privacy to reduce privacy risks in real applications, such as recommendation systems and knowledge graphs.

- Conducted high-quality and impactful research: **published two papers** in top conferences, namely, ACM CCS and USENIX Security, and one under review, **received over 60 citations** since 2021 and delivered **more than 6 invited talks** at top institutions and companies, such as Imperial College London, University of Illinois, and Twitter.
- **Implemented locally private GNN models** using PyTorch-Geometric that enables online social networks to learn from their users' data without compromising their privacy. **Received over 38 stars and 13 forks** on GitHub.

- **Developed a differentially private GNN model** using PyTorch-Geometric, Opacus, and Auto-DP that enables privacy-preserving training and inference of GNNs. **Received over 20 stars and 1 fork** on GitHub.
- **Finalist** in CSAW Applied Research Competition for the best paper award in computer security in Europe.
- **Offered a travel grant** to attend CISPA Summer School 2022 on Trustworthy Artificial Intelligence in Germany.

Visiting Collaborator
The Alan Turing Institute, London, UK, March 2023

Joined the Safe and Ethical AI group as a visiting PhD student to contribute to their ongoing research on trustworthy AI.

- **Co-organized a workshop** on Privacy and Fairness in AI for Health with **11 invited speakers** from top research institutes and companies, such as Oxford, Microsoft, and DeepMind, and **over 60 attendees**.

Research Intern
Brave Software, San Francisco, CA, USA (Remote), Mar 2022 – May 2022

Recruited as a member of the research team to contribute to privacy-preserving machine learning research, aiming to improve Brave Browser's ads and news recommendation systems.

- **Hand-picked to work on a project** about federated reinforcement learning for privacy-preserving recommendation.
- **Developed an experimental framework** to evaluate the performance of federated neural bandits under client heterogeneity using PyTorch and Flower for server-side simulation and Tensorflow-Lite for mobile clients.

Data Engineer
Sharif ICT Innovation Center, Tehran, Iran, Sep 2018 – May 2019

Served in a part-time role as a member of an R&D team to develop a native big-data processing platform for the center.

- **Conducted a comprehensive study on massively scalable graph databases**, such as Neo4j, OrientDB, and JanusGraph, by setting up, configuring, and benchmarking them on a computing cluster.
- **Developed a dashboard** to discover key insights from customers' data using Kibana, Elasticsearch, and Cassandra.

Research Assistant
Sharif University of Technology, Tehran, Iran, Nov 2014 – May 2019

Joined the Data Science and Machine Learning Lab as a master's student and continued as a research assistant after graduation. Worked on a range of research projects in the areas of privacy-preserving machine learning, web data science, and social and information network analysis.

- Conducted high-quality and impactful research: **published four papers** in top-tier venues, including IEEE IoTJ, ACM TKDD, and TheWebConf, **with over 350 citations** since 2016.
- Worked on a hybrid mobile-server learning architecture based on Siamese fine-tuning and split learning to make non-private pre-trained deep learning models privacy-preserving at the inference stage.
- **Conducted large-scale data analysis** using a collection of recipes published on the web and their content, aiming to understand cuisines and culinary habits around the world. **Received media coverage** from prominent news outlets, such as MIT Technology Review, The Independent, and France 24.
- **Developed time-aware link prediction algorithms** over heterogeneous social networks using recurrent neural networks and non-parametric machine learning.
- Received **research assistantship** from IBM-ILLINOIS Center for Cognitive Computing Systems Research (declined due to visa issues).

Teaching Experience

Lecturer
International AI Doctoral Academy (AIDA), Online, Nov 2022

Course: An Introduction to Trustworthy Machine Learning, November 2022

Website: https://www.i-aida.org/course/an-introduction-to-trustworthy-machine-learning/

Lecturer
Sharif University of Technology, Tehran, Iran, Fall 2017

Course: Fundamentals of Programming (Python), Fall 2017

Website: http://ce.sharif.edu/courses/96-97/1/ce153-12/

Teaching Assistant
Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland,

Courses: Computational Social Media (Spring 2021, 2022, and 2023)

Teaching Assistant
Sharif University of Technology, Tehran, Iran,

Courses: Artificial Intelligence (Spring 2017), Advanced Topics in Artificial Intelligence - Statistical Learning Theory (Spring 2016), Engineering Probability and Statistics (Spring 2016)

Teaching Assistant
University of Isfahan, Esfahan, Iran,

Courses: Artificial Intelligence (Fall 2013), Advanced Computer Programming 2 - JavaFx and Android (Fall 2012), Computer Programming - Java (Fall 2011), Computer Programming - C++ (Fall 2010)

Community and Professional Services

Invited Speaker
Imperial College London (2023, 2020), University of Illinois at Chicago (2022), L3S Research Center (2022), Graph Neural Networks User Group Meetup (2021), Twitter Machine Learning Seminar (2021)

Organizing Committee Member
Privacy and Fairness in AI for Health Workshop (2023)

Program Committee Member
ACM Workshop on Wireless Security and Machine Learning (WiseML) (2023), ICLR PAIR2Struct Workshop (2022)

Scientific Peer Reviewer
Learning on Graphs Conference (2023, 2022), International Conference on Artificial Intelligence and Statistics (AISTATS) (2023), Artificial Intelligence Journal (2022), IEEE Transactions on Big Data (2021), ICLR Workshop on Distributed and Private Machine Learning (2021), ACM Transactions on Intelligent Systems and Technology (2020), Social Network Analysis and Mining Journal (2020), World Wide Web Journal (2018)

Publications

[1] Sina Sajadmanesh and Daniel Gatica-Perez
ProGAP: Progressive Graph Neural Networks with Differential Privacy Guarantees
Technical Report, ArXiv e-prints, Apr 2023

[2] Sina Sajadmanesh, Ali Shahin Shamsabadi, Aurélien Bellet, and Daniel Gatica-Perez
GAP: Differentially Private Graph Neural Networks with Aggregation Perturbation
USENIX Security Symposium (USENIX Security 23), Aug 2023

[3] Sina Sajadmanesh and Daniel Gatica-Perez
Locally Private Graph Neural Networks
ACM Conference on Computer and Communications Security (CCS 2021), Nov 2021

[4] Sina Sajadmanesh and Daniel Gatica-Perez
When Differential Privacy Meets Graph Neural Networks
Technical Report, ArXiv e-prints, Jun 2020

[5] Seyed Ali Osia, Ali Shahin Shamsabadi, Sina Sajadmanesh, *et al.*
A Hybrid Deep Learning Architecture for Privacy-Preserving Mobile Analytics
IEEE Internet of Things Journal, May 2020

[6] Sina Sajadmanesh, Sogol Bazargani, Jiawei Zhang, and Hamid R. Rabiee
Continuous-Time Relationship Prediction in Dynamic Heterogeneous Information Networks
ACM Transactions on Knowledge Discovery from Data, Aug 2019

[7] Sina Sajadmanesh, Jiawei Zhang, and Hamid R. Rabiee
NPGLM: A Non-Parametric Method for Temporal Link Prediction
Technical Report, ArXiv e-prints, Jun 2017

[8] Sina Sajadmanesh, Sina Jafarzadeh, Seyed Ali Ossia, *et al.*
Kissing Cuisines: Exploring Worldwide Culinary Habits on the Web
International World Wide Web Conference (WWW 2017) Companion, Apr 2017

[9] Sina Sajadmanesh, Hamid R. Rabiee and Ali Khodadadi
Predicting Anchor Links between Heterogeneous Social Networks
IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, Aug 2016