TOWARDS EFFICIENT SELF-SUPERVISED LEARNING ON GRAPHS

A Dissertation

by

QIAOYU TAN

Submitted to the Graduate and Professional School of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Xia (Ben) Hu
Committee Members,	Shuiwang Ji
	Nick Duffield
	Ruihong Huang
Head of Department,	Scott Schaefer

August 2023

Major Subject: Computer Engineering

ABSTRACT

Deep learning on graphs has garnered considerable attention across various machine learning applications, encompassing social science, transportation services, and biomedical informatics. Nonetheless, prevailing methods have predominantly focused on supervised learning, resulting in several limitations, such as heavy reliance on labels and subpar generalization.

To address the scarcity of labels, self-supervised learning (SSL) has emerged as a promising approach for graph data. Traditional SSL methods for graphs primarily concentrate on enhancing model performance through advanced data augmentation strategies and contrastive loss functions. Despite the significant progress made by existing studies, they encounter severe efficiency challenges when dealing with large-scale graphs and resource-limited applications, such as online services. To bridge this gap, I have developed a series of graph SSL models that systematically enhance the efficiency of self-supervised learning on graphs across the stages of model training, inference, and deployment. Firstly, to improve training efficiency, we propose automating the data augmentation process through Graph Personalized Augmentation (GPA) and conducting augmentation-free training via model perturbation (PerturbGCL). Secondly, to expedite inference efficiency, we suggest distilling the fine-tuned classification model into a lightweight model using reliable knowledge distillation (Meta-MLP). Finally, to enhance deployment efficiency, we propose the development of a universal graph model (S2GAE) that enables the learned representation to generalize across different types of downstream tasks in the graph system.

My research presents a significant contribution to the research community by advancing the efficiency and applicability of self-supervised learning on graphs, addressing challenges related to label scarcity and resource limitations. These innovations have the potential to revolutionize various machine learning applications across disciplines, ranging from social science to transportation services and biomedical informatics, ultimately paving the way for more effective and widespread adoption of deep learning techniques in real-world graph scenarios.

ACKNOWLEDGMENTS

Throughout the writing of this thesis, I have received an immense amount of support and assistance. I would like to take this opportunity to express my sincere thanks and appreciation to all those who have contributed to my research and the completion of this thesis.

First and foremost, I would like to express my heartfelt gratitude to my advisor, Dr. Xia (Ben) Hu, for his invaluable guidance, mentorship, and unwavering support throughout my Ph.D. journey. Dr. Hu not only imparted his profound knowledge in machine learning and data mining, but also equipped me with essential research skills, including scholarly paper writing, topic selection, and the development of independence as a researcher. His support, encouragement, and constructive criticism have played a vital role in shaping my academic and research career.

Secondly, I would like to express my sincere appreciation to my esteemed Ph.D. committee members, Dr. Shuiwang Ji, Dr. Nick Duffield, and Dr. Ruihong Huang, for their insightful feedback, constructive criticism, and valuable suggestions. Their expertise and critical evaluations have significantly enriched the quality of this work.

Also, I would like to extend my gratitude to all the members of my research group at the DATA Lab, with whom I have had the pleasure of collaborating and exchanging ideas. Their valuable insights and constructive feedback have been indispensable in advancing my research and overcoming various challenges.

More importantly, I wish to express my deepest appreciation to my family members for their unwavering love and understanding throughout this journey. Their constant encouragement and belief in my abilities have served as an enduring source of inspiration and motivation.

Finally, I would like to express my gratitude to Texas A&M University and all the funding agencies, including the National Science Foundation and the Defense Advanced Research Projects Agency, for their support and financial assistance.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by the dissertation committee consisting of Professor Xia Hu [advisor], Professor Shuiwang Ji and Professor Ruihong Huang from the Department of Computer Science and Engineering, and Professor Nick Duffield from the Department of Electrical and Computer Engineering.

All the work conducted for the dissertation was completed by the student independently.

Funding Sources

This work is, in part, supported by DARPA (#N66001-17-2-4031, #W911NF-16-1-0565), and NSF (#IIS-1849085, #IIS-1750074, #IIS-2006844). The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

1.1 Background and Motivation

Graphs have become the *de facto* natural language to represent objects and their relations across various domains, including social networks [7, 8, 9], knowledge base [10, 11], recommendation systems [12, 13, 14, 15, 16, 17], and molecules [18, 19, 20]. Within the realm of artificial intelligence research, graph neural networks [21, 22, 23, 24] (GNNs) have gained significant popularity as an effective deep learning paradigm for modeling graph-structured data. The majority of GNN studies have focused on (semi-)supervised learning scenarios, wherein a substantial volume of task-specific labels is utilized for training models. However, the reliance on labels poses limitations in numerous real-world applications due to factors such as label scarcity [25, 26, 27], high costs [28, 29, 30], label imbalance [31, 32, 33], label noise [34], or even the absence of labels [35, 36].

To address this challenge, self-supervised learning [37, 38, 39, 40] (SSL) has emerged as a promising paradigm to alleviate the dependence on manual labels. SSL offers a learning framework where representations are learned directly from unlabeled data. In scenarios where a limited number of labeled data is available, SSL can serve as a pre-training stage, followed by fine-tuning the pre-trained deep models using the labeled data for downstream tasks. This approach reduces the reliance on extensive labeled data and opens avenues for leveraging large amounts of unlabeled data to improve model performance.

Recognized as “the key to human-level intelligenc" by Turing Award winners Yoshua Bengio and Yann LeCun, self-supervised learning (SSL) has recently made significant strides in computer vision (CV) and natural language processing (NLP). Early SSL studies mainly focus on designing various semantics-related pretext tasks, such as image inpainting [41], image colorizing [42], and jigsaw puzzle [43] for images, as well as masked language model [44] for texts, to facilitate representation learning. Currently, contrastive learning-based SSL frameworks, such as MoCo [45],

SimCLR [46], and BYOL [47], have emerged as state-of-the-art approaches for learning visual features by exploiting the invariance of semantics under data augmentation.

Building upon the remarkable achievements of SSL in CV and NLP, there is a growing interest in extending SSL to graph-structured data. For example, GraphCL [48] introduced various straightforward data augmentation strategies for graphs and extended the concept of SimCLR to GNN pre-training. GCC [49] proposed a subgraph sampling-based contrastive method inspired by MoCo. Additionally, BGRL [1] developed a negative-free contrastive learning approach for graphs, drawing inspiration from BYOL.

Despite the plethora of graph SSL methods proposed in recent years, their primary focus lies in enhancing model performance through advanced data augmentations and pre-training loss objectives. However, the efficiency aspect of GNN pre-training has been largely neglected, impeding the practicality of self-supervised GNN models in real-world applications, particularly when confronted with resource-constrained scenarios and large-scale graphs. Motivated by this research gap, I aim to enhance the efficiency of SSL on graphs by addressing the following research questions. Firstly, how can we efficiently select the most informative augmentations for each instance in a graph dataset from a pool of augmentation candidates, thereby eliminating the need for laborious trial-and-error procedures? Secondly, given the time-consuming and expensive nature of graph augmentation itself, is it possible to expedite the training of self-supervised graph models without explicitly conducting data augmentation? Thirdly, apart from training efficiency, pre-trained GNN models inherently face inference challenges as they require collecting neighbors multiple hops away from the anchor node for effective message propagation. Therefore, can we distill the knowledge of pre-trained GNN models into lightweight neural networks, enabling efficient deployment in online scenarios while retaining the effectiveness of GNN? Furthermore, considering that real-world graph systems often necessitate the provision of multiple graph services concurrently, can we enhance the generalization capability of pre-trained GNN models to effectively address various downstream tasks using a unified model?

Below, we provide a detailed analysis of each research question.

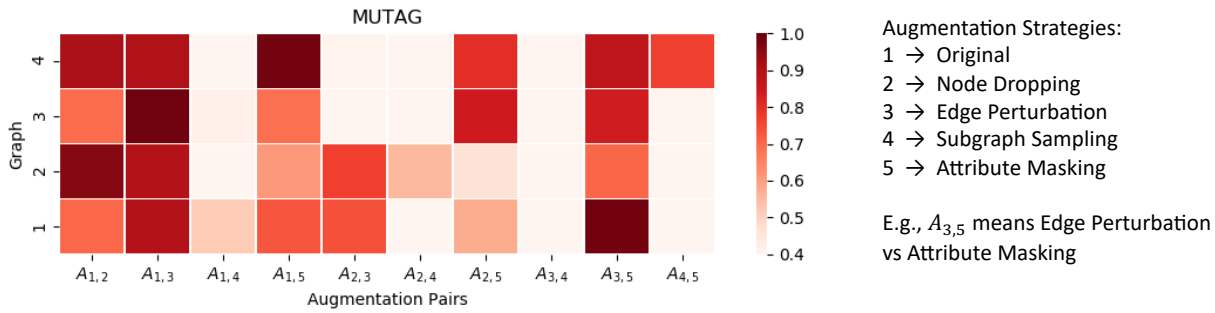

Figure 1.1: The effect of different augmentation strategies toward four randomly sampled graphs from MUTAG. X-axis denotes the id of augmentation pair. Y-axis is the graph id. The color represents the performance. The darker the color is, the better performance GCL achieves under the corresponding augmentation strategy.

1.1.1 The Training Efficiency Challenge Arises from the Personalized Augmentation

The performance of GCL is known to be heavily affected by the chosen augmentation types [50, 48], since different augmentations may impose different inductive biases about the data. Intensive recent works have been devoted to exploring effective augmentations for different graph scenarios [51, 52, 53]. Typical augmentation strategies include node dropping, edge perturbation, subgraph sampling, and attribute masking. The best augmentation option is often data-driven and varies in graph scales or types [48]. For example, [48] revealed that edge perturbation may benefit social networks but hurt biochemical molecules. Therefore, manually searching augmentation strategies for a given scenario would involve extensive trials and efforts, hindering the practical usages of GCL. Several studies have been proposed to address this issue by automating GCL.

Despite the recent advances, existing GCL might be suboptimal for augmentation configuration, since they apply a well-chosen but identical augmentation option to all graphs in a dataset. The rationale is: graphs in a scenario usually have different properties because the characteristics of real-world graphs are complex and diverse [54, 55]. For example, by slightly changing the structure of a molecular graph, its target function could be completely different [56]. Similar observations have also been found in many other graph domains, such as social communities and protein-protein interaction networks [1]. Motivated by these observations, we conduct a pre-

liminary experiment on the MUTAG dataset to test how different augmentation types impact the GCL results. Results in Fig. 1.1 show that different graphs favor distinct perturbation operations to achieve their best performance. For example, graph 1 performs better when using *edge perturbation* vs *attribute masking*, while graph 2 prefers *original* vs *node dropping*. Such personalized phenomenon of graph instances, in terms of their desirable augmentation strategies, poses significant challenge in selecting the optimal augmentation strategies, since the personalized search space becomes exponential to the number of graphs.

1.1.2 The Training Efficiency Challenge Arises from Graph Augmentation

To pre-train GNN models, existing studies mainly focus on generating augmented graphs based on data augmentations. However, finding desirable augmentations requires cumbersome efforts, as the optimal augmentations are domain-specific and vary from graph to graph [48]. While some efforts have been devoted to accelerating the search process in an automated fashion [53, 57, 58], conducting data augmentation during training is already expensive in itself, especially for complex, dense, and large-scale graphs. This is because advanced data augmentation strategies are random functions designed for individual nodes in the graph, for example, masking the attribute features of each node or interfering with the local neighborhood structure of the anchor node. This local interference with the graph is time-consuming when the input graph is dense and large-scale. What's worse, to avoid augmentation bias, data augmentation is often repeatedly performed per epoch or iteration of the training process, which exacerbates the efficiency challenge of model optimization, since the total number of epochs for empirical training could be hundreds or even thousands [1].

To improve the training efficiency of GCL, the intuitive idea is to avoid perturbing graph data. Inspired by this, AFGRL [59] studies contrastive learning without augmentation on graphs. Instead of interfering with the input graph to generate contrastive samples, AFGRL suggests discovering positive and negative pairs in the latent space directly. Nevertheless, it requires clustering nodes during training (e.g., k-means clustering and k-nearest neighbor search), which is still training costly. Recently, SimGRACE [2] proposed an alternative contrastive learning solution by adding Gaussian noise to the model weights of GNN branches. By only disturbing model weights,

SimGRACE achieves state-of-the-art training acceleration. However, it may lead to subpar performance because it is completely data-agnostic and limited to learning representations that are invariant to structural perturbations, which is crucial for self-supervised learning on graphs [60]. Therefore, we need an alternative contrastive learning approach to pre-train GNN models without explicitly performing graph augmentation, while maintaining model prominence.

1.1.3 The Inference Efficiency Challenge Arises from Neighborhood Collection

After a GNN model is well fine-tuned for specific downstream task, such as node classification, it still facing several challenges during inference, especially when going deeper [46, 61] and applying to large-scale graphs [62, 63]. The major reason [64] is that the message propagation among neighbors from multi-hops always incurs heavy data dependency, causing substantially computational costs and memory footprints. Some preliminary efforts attempt to fill the gap from different aspects. For example, [65] proposes to accelerate inference via model pruning, and [66] suggests to directly reduce computational costs by weight quantization. Although they can speed up GNNs to some extent, the improvements are rather limited, since the data dependency issue remains unresolved. Recently, GLNN [5] tries to tackle this issue by compressing GNNs to inference-friendly multi-layer perceptrons (MLPs) via knowledge distillation (KD). Similar to standard KD protocols [67], GLNN trains the MLP student by using the soft labels from GNN teacher as guidance, and then deploys the distilled MLP student to conduct latency-constrained inference.

However, directly leveraging soft labels from the GNN teacher is suboptimal when the labeled nodes are scarce, a common scenario in graph-structured data [21, 68, 69]. This is mainly because a large portion of unlabeled nodes will be incorrectly predicted by GNNs due to its limited generalization ability. For instance, many GNN variants [21, 70, 71] can achieve 100% accuracy on the training set, yet their test accuracy is merely around 80% on Planetoid benchmarks. As a result, the soft labels of those wrongly predicted unlabeled nodes would introduce noises to the optimization landscape of the MLP student, leading to an obvious performance gap w.r.t. the GNN teacher [5]. It thus would be more desirable if we are able to filter out those unreliable soft labels out before training the MLP student.

1.1.4 The Deployment Efficiency Challenge Arises from Limited Model Capacity

Self-supervised generative models, exemplified by MAE [72] and BERT [44], have demonstrated remarkable performance in acquiring generalizable representations in various domains such as computer vision [73, 74] and natural language processing [75]. Such representations offer the advantage of being easily adaptable to diverse downstream tasks. In graph domains, generalizable representations also hold significant value in real applications, such as social network platforms where we may want to conduct recommendation [76] (link prediction), community detection [77] (node clustering) and malicious account detection [78] (node classification) simultaneously. Thus, generalizable node representations are desirable.

However, we have observed that it is challenging for existing self-supervised generative models on graphs to meet the above expectation. To date, no self-supervised graph studies have succeeded in performing comparable results with GCL methods on node-level and graph-level classification scenarios without sacrificing their promise on link prediction tasks. This phenomenon casts doubt on the generalizability of self-supervised graph models as a universal graph learner, leading to serious deployment challenges as we need to design and maintain multiple graph models for different downstream tasks within the same graph system. Consequently, we may want to know why traditional self-supervised graph models can not generalize well to graph classification tasks? How to build a generalizable self-supervised graph framework that could perform well on link-level, node-level, and graph-level learning tasks simultaneously?

1.2 Thesis Contributions

To address the aforementioned challenges, this thesis introduces a series of efficient self-supervised learning (SSL) algorithms, aiming to explore the following research questions: (Q1) Can we enable personalized augmentation strategies for each graph in a given scenario, facilitating the identification of effective augmentations based on their unique characteristics? (Q2) Is data augmentation indispensable for graphs? Can we develop an efficient SSL method based solely on model perturbation, eliminating the need for explicit data augmentation? (Q3) How can we per-

form latency-constrained SSL on graphs to enable the deployment of learned models in real-world scenarios? (Q4) How can we enhance the generalization capability of self-supervised graph models, ensuring their efficacy across a diverse range of downstream tasks? The key contributions can be summarized as below.

- The first contribution (**C1**) involves the development of an efficient personalized augmentation selector for graph classification tasks. We begin by defining the personalized augmentation problem in graphs and highlighting the challenge of searching through an extensive search space. Subsequently, we introduce a plug-and-play personalized augmentation module, which can be seamlessly integrated with various graph SSL algorithms in an end-to-end manner.

- Recognizing the time-consuming and risky nature of conducting data augmentation on graphs, the second contribution (**C2**) explores an alternative approach to achieve effective SSL on graphs. Our focus lies in model perturbation for representation learning. Instead of searching for optimal combinations to perturb nodes, edges, or attributes, we propose conducting perturbation on the model architectures themselves, specifically Graph Neural Networks (GNNs).

- To expand the applicability of SSL on graphs to resource-constrained scenarios like online serving and edge devices, we delve into employing model compression techniques on well-trained SSL models. The third contribution (**C3**) entails the development of a novel knowledge distillation framework for pre-trained GNNs, specifically targeting practical and challenging few-shot learning scenarios. We propose a reliable knowledge distillation algorithm that effectively transfers knowledge from a GNN teacher to a pure MLP student by leveraging both labeled and unlabeled samples.

- Given the numerous graph learning tasks in industrial applications and their shared characteristics, the fourth contribution (**C4**) of this thesis focuses on foundational settings and delves into the development of a universal SSL algorithm for graphs. To accomplish this, we propose a self-supervised Graph Autoencoder (GAE) framework that harnesses the potential of GAEs with minimal, yet meaningful, endeavors. In particular, rather than reconstructing the entire input

structure, we employ a random masking technique to conceal a portion of edges. Subsequently, we aim to learn the reconstruction of these missing edges through the utilization of an effective masking strategy and an expressive decoder network.

1.3 Related Work

The work in this thesis covers several research topics, including graph neural networks (GNN), self-supervised learning, and GNN acceleration.

1.3.1 Graph Neural Networks

With the rapid advancement of graph neural networks (GNNs), a plethora of GNN-based graph representation learning frameworks have been proposed [79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90], demonstrating promising performance. These methods can typically be categorized into supervised and unsupervised approaches. While supervised methods [91, 92, 23, 93] achieve notable results by leveraging labeled data, real-world scenarios often suffer from limited availability of reliable labels. Consequently, unsupervised graph learning approaches [94, 95, 22, 6] possess broader applicability. For instance, the well-known method GAE [94] learns graph representations by reconstructing the network structure within an autoencoder framework. Another popular approach, GraphSAGE [22], trains GNNs through a random-walk-based objective.

1.3.2 Self-Supervised Learning

Inspired by the achievements in self-supervised learning for image and text data, researchers have made extensive efforts to explore self-supervised learning on graphs. These studies can be categorized into two main approaches: graph contrastive learning and graph generative learning. Graph contrastive learning aims to learn node representations by pulling together representations of related objects while pushing apart unrelated ones [50, 52]. For instance, some works [51, 96] propose maximizing the mutual information between node-level and graph-level representations. Others [50, 52] suggest maximizing agreement between two augmented views generated by specific data augmentations [97]. These methods have demonstrated promising results in classification tasks, including node classification [98, 52, 3] and graph classification [48, 99, 58]. However, their

performance heavily relies on informative augmentation strategies [100], which can be challenging to achieve due to the complexity of interpreting perturbed graphs [6]. Moreover, complicated training strategies such as momentum update [49] or stop-gradient [1] are often required to stabilize the training.

On the other hand, the graph autoencoder is a classical generative framework used for representation learning on graphs. The earliest works can be traced back to DeepWalk [101] and Node2vec [102], which design the encoder as a simple embedding lookup table and utilize random-walk objectives [55] to train the model. More recently, work in GAE [94, 22] suggests adopting GNNs [21] as encoder and simplifying the reconstruction target as the input graph structure [103]. Subsequently, numerous studies have focused on leveraging the network structure (e.g., ARVGA [103]) or incorporating additional side information [104, 105, 106]. Currently, GAEs have become the de facto standard for fine-grained graph analytical tasks such as link prediction in the literature. Moreover, masked graph autoencoders [6, 107, 108] have recently garnered significant attention. Instead of reconstructing the observed information in the original graphs, these methods involve masking a portion of the input data and subsequently learning to recover the masked content for GNN pre-training.

1.3.3 GNN Acceleration

Due to the nature of message propagation in GNN models, there has been significant interest in accelerating GNN models. Existing efforts for GNN speedup can be broadly categorized into two main approaches: scalable training and inference acceleration. Scalable training focuses on scaling GNNs to handle large-scale graphs with millions or even billions of nodes. Examples of such methods include sampling-based approaches [22, 91, 63], clustering-based methods [62], and decoupling techniques [109, 110, 111]. Unfortunately, these methods often face challenges during inference due to the computational complexity of the message propagation process [112].

Inference acceleration, on the other hand, aims to reduce the inference latency of GNNs to extend their applicability in resource-constrained applications. Initial attempts have been made using pruning-based [65, 61, 113] and quantization-based [114] techniques. However, the improvements

achieved by these methods are limited as they still rely on message propagation for embedding.

More recently, knowledge distillation (KD) has emerged as an alternative approach for accelerating GNN inference. However, most existing methods focus on distilling large GNNs into smaller ones [115, 116, 117, 118, 119] or propagating labels to student models [120]. Since message propagation is still required in these approaches, they encounter the same inference issues as standard GNNs. To address this limitation, a recent work called GLNN [5] aims to accelerate GNN inference by distilling it into a lightweight MLP student.

2. EFFICIENT GRAPH CONTRASTIVE TRAINING WITH AUTOMATED AUGMENTATION SELECTION

The training process of graph contrastive learning (GCL) has faced challenges when it comes to personalized augmentation for large-scale graphs. However, efficiently performing personalized augmentations in GCL is difficult due to two main reasons. First, unlike traditional GCL settings, the search space in personalized scenarios grows exponentially with the number of graphs (N) in a dataset. This vast search space becomes impractical when N reaches thousands or even tens of thousands [121]. Consequently, the conventional trial-and-error approach is not feasible because each trial, which involves testing an augmentation option for a dataset, is time-consuming. Second, the choice of augmentations and the GCL model have a mutually reinforcing relationship. The contrastive loss in GCL comprises augmented views and the GCL encoder. In other words, improving GCL performance requires well-chosen augmentation strategies [48], while selecting suitable augmentation operators relies on the signals provided by GCL as feedback [53]. Therefore, performing effective personalized augmentation selection while considering this mutual effect poses another challenge.

To address these challenges, we propose a novel contrastive learning framework called Graph Personalized Augmentation (GPA). Our approach aims to investigate two research questions: 1) What are the impacts of different augmentation strategies on a given graph instance within a graph dataset? 2) Can we construct a stronger automated GCL model by allowing each graph instance to choose its preferred augmentation types? GPA involves iteratively updating a personalized augmentation selector and the GCL method. The former identifies the optimal augmentation types for each graph instance, while the latter is trained based on an instance-level contrastive loss defined using the assigned augmentation options.

2.1 Problem Statement

Let $\mathcal{G} = \{\mathcal{G}_n : 1 \leq n \leq N\}$ denote a graph set with N sample graphs, where $\mathcal{G}_n = (\mathcal{V}, \mathcal{E}) \in \mathcal{G}$ stands for an undirected graph with nodes $\mathcal{V}$ and edges $\mathcal{E}$. Each node $v \in \mathcal{V}$ in $\mathcal{G}_n$ is described by an F-dimensional feature vector $\mathbf{X}_v \in \mathbb{R}^F$. We use $\mathcal{A} = \{A_k : 1 \leq k \leq K\}$ to denote a set of data augmentation operators, where K is the maximum number of augmentation types of interest. Each augmentation operator $A_k : \mathcal{G}_n \to \tilde{\mathcal{G}}_n$ transforms a graph into its conceptually similar form with certain prior. In previous GCL studies, they focus on identifying two optimal augmentation types for the whole dataset $\mathcal{G}$, such as the augmentation pair $A_{i,j} = (A_i, A_j)$, where $A_i, A_j \in \mathcal{A}$. The optimal augmentation pair here is often manually picked via rules of thumb or trial-and-error. However, as shown in Fig. 1.1, different graphs within the dataset may favor different augmentation combinations. Therefore, we study personalized augmentation selection and formally define the research problem as below.

Definition 1. ***Personalized augmentation selection.*** *Given a set of graphs $\mathcal{G} = (\mathcal{G}_n : 1 \leq n \leq N)$, and the augmentation space $\mathcal{A} = \{A_k : 1 \leq k \leq K\}$ consisting of K different augmentation types, to perform GCL, personalized augmentation selection aims to find the optimal augmentation pair $A^n_{i,j} = (A^n_i, A^n_j)$ for each graph $\mathcal{G}_n \in \mathcal{G}$. The values of i and j are only determined by the characteristic of the n-th sample graph $\mathcal{G}_n$.*

2.2 Methodology

In this section, we present the details of the proposed GPA shown in Fig. 2.1. In a nutshell, it contains two critical components: the *personalized augmentation selector* and the *GCL model*. The former module aims to infer augmentation choices for the downstream GCL methods when training them on the training set, while the later provides reward to update the augmentation selector based on the validation set. In the following, we first illustrate the exponential selection space of our personalized augmentation setting. Then, we elaborate the details of the augmentation selector and the GCL method. Finally, we show how to jointly optimize the two components in a unified perspective.

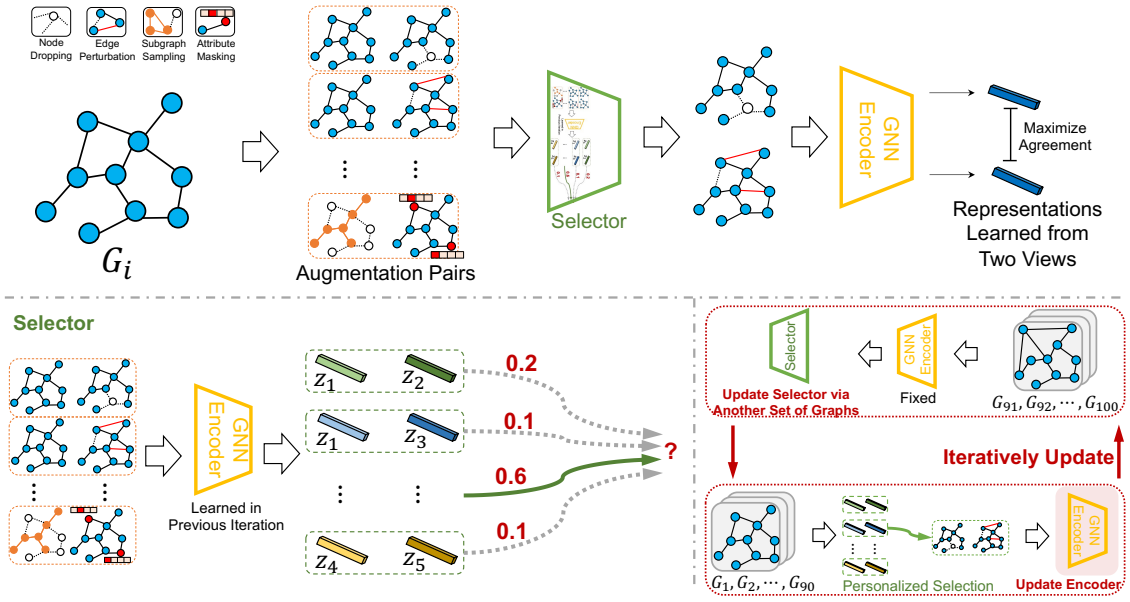

Figure 2.1: Illustration of our GPA framework. The *personalized augmentation selector* infers the two most informative augmentation operators, and the *GCL model* trains GNN encoder based on the sampled augmented views. Specifically, the *personalized augmentation selector* is learned to adjust its selection strategy to infer optimal augmentations on each graph, according to the characteristic of each graph and the *GCL model*'s performance, i.e., loss.

2.2.1 Personalized Augmentation Selection Space

Given the graph dataset $\mathcal{G} = \{\mathcal{G}_n : 1 \leq n \leq N\}$ and a pool of augmentation operators $\mathcal{A} = \{A_1, A_2, \cdots, A_K\}$, existing GCL efforts aim to select two informative operators (e.g., $(A_i, A_j \mid 1 \leq i, j \leq K)$) for N graphs to create their augmented views. Since the augmentation operators are shared for the whole dataset, the total selection space is $\binom{K+1}{2}$, i.e., sampling two operators with replacement. This collective selection strategy is widely adopted in existing GCL works. However, as discussed before, various sample graphs may favor different augmentation operators owing to the diversity of graph-structured data. Therefore, we propose to adaptively choose two augmentation strategies for different graphs. Following Definition 1, we define the selected augmentations for each graph $\mathcal{G}_n$ as $A^n_{i,j} = (A^n_i, A^n_j)$. Then, the potential augmentation selection size for each graph is $\binom{K+1}{2}$, and the total selection space for the whole dataset equals to $\binom{K+1}{2}^N$. Although K is empirically small (e.g., $K = 5$) in GCL domain, the total selection space

in our personalized setting is still huge and intractable, since the complexity grows exponentially to the number of graphs. For instance, when $K = 5$ and $N = 100$, we already have 15^{100} selection configurations roughly. The situation is more serious in real-world scenarios where N is thousands or even tens of thousands. In this paper, we adopt five essential augmentation operators (i.e. $K = 5$), denoted by $\mathcal{A} = \{\textbf{Identical}, \textbf{NodeDrop}, \textbf{EdgePert}, \textbf{Subgraph}, \textbf{AttMask}\}$. The augmentation pairs are shown in Table 2.1, and the details of these augmentations are listed below.

- **NodeDrop.** Given the graph $\mathcal{G}_n$, NodeDrop randomly discards a fraction of the vertices and their connections. The dropping probability of each node follows the i.i.d. uniform distribution. The underlying assumption is that missing part of vertices does not damage the semantic information of $\mathcal{G}_n$.

- **EdgePert.** The connectivity in $\mathcal{G}_n$ is perturbed through randomly adding or dropping a certain portion of edges. We also follow the i.i.d. uniform distribution to add/drop each edge. The underlying prior is that the semantic meaning of $\mathcal{G}_n$ is robust to the variance of edges.

- **AttMask.** AttMask masks the attributes of a certain proportion of vertices. Similarly, each node's masking possibility follows the i.i.d. uniform distribution. Attribute masking implies that the absence of some vertex attributes does not affect the semantics of $\mathcal{G}_n$.

- **Subgraph.** This augmentation method samples a subgraph from the given graph $\mathcal{G}_n$ based on random walk. It believes that most of the semantic meaning of $\mathcal{G}_n$ can be preserved in its local structure.

In summary, by considering personalized augmentation, the search space per dataset increases from $\binom{K+1}{2}$ to $\binom{K+1}{2}^N$. Therefore, common selection techniques such as rules of thumb or trial-and-errors adopted by prior GCL approaches [53, 48] are no longer appropriate. Thus, a tailored augmentation selector is needed to effectively tackle the challenging personalized augmentation problem.

Table 2.1: Distinct graph augmentation pairs.

$A_{i,j}$	A_i, A_j	Augmentation	Augmentation
$A_{1,1}$	A_1,A_1	Identical	Identical
$A_{1,2}$	A_1, A_2	Identical	Node Dropping
$A_{1,3}$	A_1, A_3	Identical	Edge Perturbation
$A_{1,4}$	A_1, A_4	Identical	Subgraph
$A_{1,5}$	A_1, A_5	Identical	Attribute Masking
$A_{2,2}$	A_2, A_2	Node Dropping	Node Dropping
$A_{2,3}$	A_2, A_3	Node Dropping	Edge Perturbation
$A_{2,4}$	A_2, A_4	Node Dropping	Subgraph
$A_{2,5}$	A_2, A_5	Node Dropping	Attribute Masking
$A_{3,3}$	A_3, A_3	Edge Perturbation	Edge Perturbation
$A_{3,4}$	A_3, A_4	Edge Perturbation	Subgraph
$A_{3,5}$	A_3, A_5	Edge Perturbation	Attribute Masking
$A_{4,4}$	A_4, A_4	Subgraph	Subgraph
$A_{4,5}$	A_4, A_5	Subgraph	Attribute Masking
$A_{5,5}$	A_5, A_5	Attribute Masking	Attribute Masking

2.2.2 Personalized Augmentation Selector

In order to assign different augmentation operators to various sample graphs when performing GCL on a specific dataset, random selection is the intuitive solution. Its key idea is to randomly sample two augmentation types for each graph from the candidate set. Despite the simplicity, the random selection approach fails to control the quality of sampled augmentation operators. Therefore, directly coupling the existing GCL framework with random augmentation selection would lead to performance degradation.

To address this issue, we focus on data-driven search by making the augmentation selection process learnable. The principle idea is to parameterize our personalized augmentation selector with a deep neural network, which takes a query graph as input and outputs its optimal augmentation choices. There are two main hurdles to achieve this goal: (i) given the exponential augmentation space, how can we make our personalized augmentation selector scale to the real-world dataset with thousands of graphs; (ii) since the topology structure and node attributes are crucial to graph-structured data, how can our personalized augmentation selector exploit this information to produce a more precise augmentation choice? We illustrate our dedicated solutions below.

Given the augmentation pool $\mathcal{A} = \{A_i : 1 \leq i \leq K\}$ and a query graph $\mathcal{G}_n$, our augmentation selector is required to select the most two informative augmentations, e.g., (A_1^n, A_3^n), from the candidate set. This selection problem is well-known to be discrete and non-differentiable. Although enormous efforts based on evolution or reinforcement learning have been proposed to address the discrete selection problem, they are still not suitable for such a large selection space (illustrated in Sec. 2.2.1) and are far from utilizing the properties of graphs. Therefore, we propose to make the search space learnable by relaxing the discrete selection space to be continuous inspired by [122] and further make this relaxation consider the characteristic of graphs. Specifically, given the sampled augmentation pair $A_{i,j}^n = (A_i^n, A_j^n)$ of $\mathcal{G}_n$, its importance score $\hat{\alpha}_{i,j}^n$ is computed as

$$\hat{\alpha}_{i,j}^n = \frac{\exp(\alpha_{i,j}^n)}{\sum_{i'j'} \exp(\alpha_{i',j'}^n)}, \alpha_{i,j}^n = g_\theta(f_w(A_i^n(\mathcal{G}_n) \parallel A_j^n(\mathcal{G}_n))), \tag{2.1}$$

where g_θ denotes the score function that takes the representations of augmented views $A_i^n(\mathcal{G}_n)$ and $A_j^n(\mathcal{G}_n)$ as input. This design enforces the score estimation to take into account the topology and node attributes of $\mathcal{G}_n$. In practice, g_θ is parameterized as a two-layer MLP with a ReLU activation function. f_w is the GNN encoder for graph representation learning. $\parallel$ indicates the concatenation operation. Through Eq. (2.1), the discrete augmentation selection process reduces to learning a score function g_θ under the consideration of graph characteristic.

After the personalized selector is well-trained, let $\alpha^n = [\hat{\alpha}_{1,1}^n, \cdots, \hat{\alpha}_{i,j}^n, \cdots, \hat{\alpha}_{K,K}^n]$ denote the vector of importance scores associated with all different augmentation pairs of the graph $\mathcal{G}_n$. The optimal augmentation choice of $\mathcal{G}_n$ can be obtained by selecting the augmentation pair with the maximum score in α^n. For example, $A^n = (A_i^n, A_j^n)$ if $\hat{\alpha}_{i,j}^n = \arg\max_{i',j'} \hat{\alpha}_{i',j'}^n$.

To summarize, the above equation provides a principled solution to our personalized augmentation setting. On one hand, it allows augmentation selection in such a large search space via the simple forward propagation of a shallow neural network, i.e., g_θ. On the other hand, it can also infer the most informative augmentations for each graph based on its own characteristic for downstream GCL model training(discussed in Sec. 2.2.3).

2.2.3 GCL Model Learning

After the personalized augmentation selector is plugged in, we can adopt it to train the GCL model. Notice that our model is applicable to arbitrary GCL methods that rely on two augmented views as input. In this section, we mainly focus on the most popular and generic GCL architecture - GraphCL [48] as the backbone and leave other specific architectures for future work.

Assume (A_i^n, A_j^n) is the optimal augmentation pair of graph $\mathcal{G}_n$ identified by our personalized augmentation selector, and $f_w(\cdot)$ is the GNN encoder. GraphCL proposes to learn $f_w(\cdot)$ by maximizing the agreement between the two augmented views, i.e., $A_i^n(\mathcal{G}_n), A_j^n(\mathcal{G}_n)$. The GNN encoder can encode the whole graph into a hidden space $\mathbb{R}^D$. In practice, a shared projection head function $\mathbb{R}^D \to \mathbb{R}^D$ is often applied upon the output of GNN encoder to improve the model capacity. In the following sections, we abuse the notation f_w to denote both the GNN encoding function and the projection function. Based on this notation, we can formally calculate the instance-level contrastive loss as follows:

$$\mathcal{L}(\mathcal{G}_n) = -\log \frac{\exp(\text{sim}(f_w(A_i^n(\mathcal{G}_n)), f_w(A_j^n(\mathcal{G}_n)))/\tau)}{\sum_{n'=1,n'\neq n}^{N} \exp(\text{sim}(f_w(A_i^n(\mathcal{G}_n)), f_w(A_j^{n'}(\mathcal{G}_{n'})))/\tau)}, \tag{2.2}$$

where $\text{sim}(\cdot,\cdot)$ denotes the cosine similarity function and τ is the temperature parameter. By minimizing Eq. (2.2), it encourages the two augmented views of the same sample graph to have similar representations, while enforces the augmented representations of disparate graphs to be highly distinct. As the sum operation over all graphs $\mathcal{G}$ in the denominator of Eq. (2.2) is computationally prohibitive, GCL is often trained under minibatch sampling [48], where the negative views are generated from the augmented graphs within the same minibatch.

Although Eq. (2.2) looks similar to the traditional contrastive loss, the key difference between them is that the augmentation operators A_i^n, A_j^n are closely related to the sample graph $\mathcal{G}_n$ in our formulation. That is, the augmentation operators learned by our model vary from one graph to another according to their own characteristics, which are previously enforced to be the same regardless of the graph's diverse nature. Benefiting from considering the graph's personality, the GCL method can learn the basic but essential features of graphs and thus achieve more expressive

representations for downstream tasks.

2.2.4 Model Optimization

Until now, we have illustrated the detailed personalized augmentation selector as well as the downstream GCL framework, the remaining question is how to effectively train the two modules. The naive solution is to first train the personalized augmentation selector separately, and then optimize the GCL method using the identified personalized augmentations as input. However, such a task-agnostic approach is sub-optimal, since it does not take the mutual reinforced effect between augmentation and the GCL method into consideration. This is because learning a better GCL method requires optimal augmentation strategies since they can augment more personalized features to distinguish it from other objects. Meanwhile, obtaining suitable augmentation strategies also needs the signals of a better GCL method as guidance for optimization. As a result, without linking the two modules in a principled way, it is almost infeasible to enforce the *personalized augmentation selector* to accurately infer optimal augmentation strategy for improving the performance of the *GCL method*. To this end, we propose to tackle this problem by jointly training the two modules under a bi-level optimization, expressed as:

$$\begin{aligned} &\arg\min_{\theta} \ \ \mathcal{L}_{valid}(w^*(\theta), \theta) \\ &s.t. \quad w^*(\theta) = \arg\min_{w} \mathcal{L}_{train}(w, \theta), \end{aligned} \tag{2.3}$$

where θ denotes the trainable parameters of the personalized augmentation selector, and w is the parameters for the GCL method. The upper-level objective $\mathcal{L}_{valid}(w^*(\theta), \theta)$ aims to find θ that minimizes the validation rewards on the validation set given the optimal w^*, and the lower-level objective $\mathcal{L}_{train}(w, \theta^*)$ targets to optimize w by minimizing the contrastive loss based on the training set with θ fixed. We want to remark that GPA only exploits the signals from the self-supervisory task itself without accessing labels. Thus, compared with conventional supervised methods, the validation set here only contains a set of graphs without label information, which is much easier to construct, e.g., randomly sampling 10% of the training set.

By optimizing Eq. (2.3), the personalized augmentation selector and the target GCL model will be jointly trained to reinforce their reciprocal effects. Since deriving exact solutions for this bi-level problem is indeed analytically intractable, we adopt the alternating gradient descent algorithm to solve it as follows.

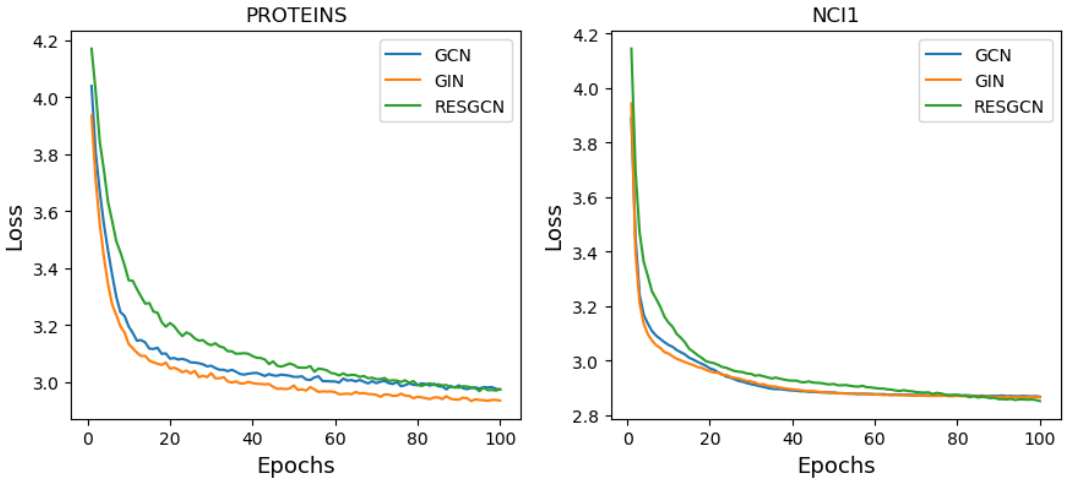

Figure 2.2: Empirical training curves of approximate gradient scheme in GPA on datasets PROTEINS and NCI1 with different GNN encoders.

2.2.4.1 Lower Level Optimization

With θ fixed, we can update w with the standard gradient descent procedure as below.

$$w' = w - \xi \nabla_w \mathcal{L}_{train}(w, \theta), \tag{2.4}$$

where $\mathcal{L}_{train} = \mathbb{E}_{\mathcal{G}_{train}} \mathcal{L}(\mathcal{G}_n)$ with $\mathcal{L}(\mathcal{G}_n)$ is instance-level contrastive loss defined in Eq. (2.2), $\mathcal{G}_{train}$ denotes the training set, and ξ is the learning rate.

2.2.4.2 *Upper Level Optimization*

Since it is not intuitive to directly calculate the gradient *w.r.t.* θ over all augmentation options, we first define the upper-level objective based on Eq. (2.2) as below:

$$\mathcal{L}_{valid}(w^*(\theta), \theta) = \sum_{A_i, A_j \in \mathcal{A}} \sum_{\mathcal{G}_n \in \mathcal{G}_{valid}} \hat{\alpha}_{i,j}^n \mathcal{L}(\mathcal{G}_n), \tag{2.5}$$

where $\hat{\alpha}_{i,j}^n$ is the selecting score computed by the score function g_θ with parameter θ defined in Eq. (2.1). $\mathcal{G}_{valid}$ denotes the validation set. Based on the above loss function, we can update θ by fixing w, expressed as:

$$\theta' = \theta - \xi \nabla_\theta \mathcal{L}_{valid}(w^*(\theta), \theta). \tag{2.6}$$

However, evaluating the gradient *w.r.t.* θ exactly is intractable and computationally expensive, since it requires solving for the optimal $w^*(\theta)$ whenever θ gets updated. To approximate the optimal solution $w^*(\theta)$, we propose to take one step of gradient descent update for w, without solving the lower-level optimization completely by training until convergence. To further compute the gradient of θ, we apply chain rule to differentiate $\mathcal{L}_{train}(w'(\theta), \theta)$ with respect to θ via w', where w' is defined in Eq. (2.4). Therefore, the gradient of θ can be approximated as:

$$\begin{aligned} \nabla_\theta \mathcal{L}_{valid}(w^*(\theta), \theta) &\approx \nabla_\theta \mathcal{L}_{valid}(w', \theta) \\ &\approx \nabla_\theta \mathcal{L}_{valid}(w - \xi \nabla_w \mathcal{L}_{train}(w, \theta), \theta). \end{aligned} \tag{2.7}$$

Applying chain rule to Eq. (2.7), we further obtain the gradient of θ as:

$$\begin{aligned} \nabla_\theta \mathcal{L}_{valid}(w^*(\theta), \theta) \approx & \nabla_\theta \mathcal{L}_{valid}(w', \theta) \\ & - \xi \nabla^2_{\theta,w} \mathcal{L}_{train}(w, \theta) \nabla_{w'} \mathcal{L}_{valid}(w', \theta) \end{aligned} \tag{2.8}$$

Algorithm 1 The framework of GPA

Input: A graph dataset $\mathcal{G}$, the personalized augmentation selector $g_\theta(\cdot)$, and a GCL model $f_w(\cdot)$;
Output: The well-trained GCL model;

```
Split the input graph dataset G into train G_train and validation G_valid set;
Initialize the selector parameter θ and the GCL model parameter w;
while not converge do
    Randomly sample a minibatch of graphs from the training set;
    Infer the optimal augmentation pairs for sampled graphs using the personalized augmentation
      selector g_θ(·);
    Update parameters w of the GCL learner based on the sampled graphs and the identified aug-
      mentation types according to Eq. (2.4);
    Randomly sample a batch of graphs from validation set;
    Compute the rewards based on the sampled validation graphs using the updated w' according
      to Eq. (2.5);
    Update parameters θ of the personalized augmentation selector according to Eq. (2.6) and
      Eq. (2.9);
Return The well-trained GCL model.
```

Since the computation cost of the second term in Eq. (2.8) is still high, it can be further approximated by the finite difference method:

$$\begin{aligned} &\xi \nabla^2_{\theta,w} \mathcal{L}_{train}(w,\theta) \nabla_{w'} \mathcal{L}_{valid}(w',\theta) \\ &\qquad \approx \frac{\nabla_\theta \mathcal{L}_{train}(w^+,\theta) - \nabla_\theta \mathcal{L}_{train}(w^-,\theta)}{2\epsilon} \end{aligned} \tag{2.9}$$

where $w^\pm = w \pm \epsilon \nabla_{w'} \mathcal{L}_{valid}(w',\theta)$ and ϵ denotes a small scalar.

By alternating the update rules in Eq. (2.4) and Eq. (2.6), we are able to progressively learn the two modules. Although an optimizer with the theoretical guarantee of convergence for the bi-level problem in Eq. (2.3) remains an open challenge, alternating gradient descent algorithm has been widely adopted to solve similar objectives in Bayesian optimization [123], Reinforcement learning [124, 125, 126, 127, 128], automatic differentiation [129, 130, 131, 132], and adversarial training [133]. The complete optimization procedure of GPA is shown in the Algorithm 5. We also show some level of empirical convergence in Figure 2.2.

2.2.5 Complexity Analysis of GPA

We analyze the complexity of GPA and illustrate its impact on the GCL backbone –GraphCL [48]. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and the GNN encoder f_w. The time complexity for the GNN backbones used in common graph learning tasks is $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$. Since the proposed GPA method is built upon GraphCL [48] and the personalized augmentation selector g_θ, the additional cost is mainly caused by the selector. For GraphCL, it performs two encoder computations per iteration plus a prediction head. If we assume the backward cost is similar to that of the forward pass, the complexity of GraphCL is $4C_{\text{encoder}}(|\mathcal{V}| + |\mathcal{E}|) + 2C_{\text{head}}(N_{\text{batch}}) + C_{\text{loss}} + 2C_{\text{aug}}$, where $C_{\cdot}$ are constants which rely on the neural architectures, and N_{batch} is the batch size. For the selector g_θ, it takes K encoder computations per iteration to generate graph representations and simple multilayer perceptrons (MLPs) to estimate the sampling probability. Thus its complexity is $KC_{\text{encoder}}(|\mathcal{V}| + |\mathcal{E}|) + C_{\text{selector}}(N_{\text{batch}}) + KC_{\text{aug}}$. Therefore, the total complexity for GPA and GraphCL are $(2K+4)C_{\text{encoder}}(|\mathcal{V}| + |\mathcal{E}|) + (2K+2)C_{\text{aug}}$ and $4C_{\text{encoder}}(|\mathcal{V}| + |\mathcal{E}|) + 2C_{\text{aug}}$, respectively, since the costs of GNN encoder and data augmentation are significantly higher than others. Assume the time cost for data augmentation is 5 times of GNN encoder, i.e., $C_{\text{aug}} = 5C_{\text{encoder}}$, then the time complexity of GPA is $\frac{2K+4+(2K+2)*5}{4+2*5} = \frac{12K+14}{14}$ times that of GraphCL. Given that $K = 5$ in our experiments, the complexity of GPA is around 6 times of GraphCL in theory. However, due to the inefficient implementation of our current code (e.g., for loop in the loss function and perform K augmentations in series), the empirical complexity of GPA is less than 7 times of GraphCL. It is noteworthy that parallel techniques such as multiprocessing could be used to accelerate the training.

2.2.6 Comparison of GPA and JOAO

While JOAO and GPA all focus on training GraphCL in an automated fashion, they differ in two crucial ways: 1) The research problem is different. JOAO aims to learn a shared sampling distribution for a given dataset, but GPA targets to learn a sampling vector for each instance (i.e., graph) in the dataset. Notably, due to randomness, JOAO can sample different augmentations to

different graphs to some extent. However, since the sampling distribution is fixed, it is "fake" personalization because it does not consider the characteristics of different graphs. In contrast, GPA could explicitly generate personalized sampling distribution to each graph, which is tailored and more challenging as the augmentation space grows exponentially to the dataset size. 2) The sampler optimization process is different. JOAO adopts a sampling-based approach to directly train a K-dimensional sampling distribution parameters. However, such a solution cannot be applied to our personalized scenarios because i) the total trainable parameters will be linear to the size of the dataset, i.e., $K \times |\mathcal{V}|$, where $\mathcal{V}$ is the dataset; and ii) it is restricted to transductive settings and cannot generate distribution vectors for new graphs, i.e., unseen graphs during training. To this end, we adopt a new approach to achieve personalized augmentation allocation by parameterizing the sampler as a neural network taking the topological and node attributes of a graph as input. It reduces the trainable parameters of the sampler to be independent with the size of the dataset.

2.3 Experiments

We evaluate the performance of GPA on multiple graph datasets with various scales and types. We focus on exploring the following research questions.

- **Q1**: How effective is GPA in performing graph representation learning against state-of-the-art GCL methods in unsupervised and semi-supervised evaluation tasks?
- **Q2**: How effective is the proposed personalized augmentation selector in identifying augmentations across various datasets?
- **Q3**: Compared with random selection, how effective is our proposed personalized augmentation selector?
- **Q4**: What are the impacts of hyperparameters on GPA, such as the embedding dimension d of the score function?

Table 2.2: Statistics of the datasets.

	$\mid \mathcal{G} \mid$	Avg.Nodes	Avg.Edges	$\#Label$
NCI1	$4,110$	29.87	32.30	2
PROTEINS	$1,113$	39.06	72.82	2
DD	$1,178$	284.32	715.66	2
MUTAG	188	17.93	19.79	2
COLLAB	$5,000$	74.49	$2,457.78$	3
IMDB-BINARY	$1,000$	19.77	96.53	2
REDDIT-BINARY	$2,000$	429.63	497.75	2
REDDIT-MULTI-5K	$4,999$	508.52	594.87	5
GITHUB	$12,725$	113.79	234.64	2
ogbg-molhiv	$41,127$	25.5	27.5	2

Table 2.3: Dataset statistics of graph-level benchmarks.

	Domain	$\mid \mathcal{G} \mid$	Avg.Nodes	Avg.Edges	#Label
NCI1	Molecule	$4,110$	29.87	32.30	2
RDT-M5K	social	$4,999$	508.52	594.87	2
GITHUB	Social	$12,725$	113.79	234.64	2

2.3.1 Datasets and Experimental Settings

Datasets. For a comprehensive comparison, we evaluate the performance of GPA on ten widely used benchmark datasets. Specifically, we include two small molecules networks (**NCI1** and **MUTAG**), two bioinformatics networks (**DD** and **PROTEINS**), and five social networks (**COLLAB**, **REDDIT-BINARY**, **REDDIT-MULTI-5K**, **IMDB-BINARY**, and **GITHUB**) from TUDatasets [56]. To evaluate the scalability of our model, we also use one large-scale OGB [121] dataset **ogbg-molhiv**. The data statistics are summarized in Table 5.2.

Learning protocols. Following common protocols [48], we aim to evaluate the performance of GPA in unsupervised and semi-supervised settings. In our model training phase, we randomly split 10% of graphs in each dataset into the validation set and use the remaining for training. After the model is trained, in unsupervised setting, we train an SVM classifier on the graph representations generated by the trained GCL model, and apply 10-fold cross-validation to evaluate the

performance. For semi-supervised setting, we finetune the GCL model (its GNN encoder) with a logistic regression layer for semi-supervised learning, where the labeled sample ratio is 0.1. To avoid randomness, we repeat the process for ten times and report the averaged results.

Implementation details. Our model is built upon Pytorch and PyG (PyTorch Geometric) library [134]. We train our model with Adam optimizer using a fixed batch size of 128. Similar to GraphCL [48], the default augmentation ratio is set to 0.2 for all augmentation types. In unsupervised setting, we adopt a three-layer GIN [135] encoder with hidden dimension 128 for all datasets. In semi-supervised scenarios, we employ a five-layer ResGCN [136] encoder with dimension 128 for TUDatasets [56], while a five-layer GIN [135] encoder with dimension 300 for OGB datasets as suggested in [121]. There is one hyper-parameter in our model, i.e., the hidden dimension d of score function g_θ. We search d within the set $\{128, 256, 512\}$. The impact of the hidden dimension is analyzed in Sec. 2.3.5.

Table 2.4: Unsupervised learning performance for graph classification in TUdatasets (Averaged accuracy $\pm$ std. over 10 runs). The **bold** numbers denote the best performance and the numbers in blue represent the second best performance

Dataset	NCI1	PROTEINS	DD	MUTAG	COLLAB	RDT-B	RDT-M5K	IMDB-B	Avg.Rank
InfoGraph	76.20 ± 1.06	74.44 ± 0.31	72.85 ± 1.78	89.01 ± 1.13	70.65 ± 1.13	82.50 ± 1.42	53.46 ± 1.03	73.03 ± 0.87	4.75
GraphCL	77.87 ± 0.41	74.39 ± 0.45	78.62 ± 0.40	86.80 ± 1.34	71.36 ± 1.15	$\mathbf{89.53 \pm 0.84}$	55.99 ± 0.28	71.14 ± 0.44	3.88
JOAO	78.07 ± 0.47	74.55 ± 0.41	77.32 ± 0.54	87.35 ± 1.02	69.50 ± 0.36	85.29 ± 1.35	55.74 ± 0.63	70.21 ± 3.08	5.00
JOAOv2	78.36 ± 0.53	74.07 ± 1.10	77.40 ± 1.15	87.67 ± 0.79	69.33 ± 0.34	86.42 ± 1.45	56.03 ± 0.27	70.83 ± 0.25	4.50
AD-GCL	69.67 ± 0.51	73.59 ± 0.65	74.49 ± 0.52	88.62 ± 1.27	73.32 ± 0.61	85.52 ± 0.79	53.00 ± 0.82	71.57 ± 1.01	5.13
AutoGCL	$\mathbf{82.00 \pm 0.29}$	75.80 ± 0.36	77.57 ± 0.60	88.64 ± 1.08	70.12 ± 0.68	88.58 ± 1.49	$\mathbf{56.75 \pm 0.18}$	73.30 ± 0.40	2.38
GPA	80.42 ± 0.41	$\mathbf{75.94 \pm 0.25}$	$\mathbf{79.90 \pm 0.35}$	$\mathbf{89.68 \pm 0.80}$	$\mathbf{76.17 \pm 0.10}$	89.32 ± 0.38	53.70 ± 0.19	$\mathbf{74.64 \pm 0.35}$	2.38

Baseline methods. To validate the effectiveness of GPA, we compare against three categories of state-of-the-art competitors. First, to evaluate the effectiveness of contrastive learning, we include one traditional network embedding method **GAE** [94]. Second, to study why we need personalized augmentation, we include classic GCL methods that assign identical augmentation strategies for all graphs **InfoGraph** [137], **GraphCL** [48], **JOAO** [53] and its variant **JOAOv2** [53].

Note that JOAO and JOAOv2 are two sample-based automated GCL methods that focus on selecting the most suitable predefined augmentation strategies for each dataset. They are the most relevant baselines to our proposed GPA. Third, we also include two view-generated automated GCL methods: **AD-GCL** [58] and **AutoGCL** [57]. Ordinarily, they should not be considered as our baselines since GPA belongs to GCL with augmentation selection, not view generation. We add them to comprehensively show the power of GPA.

Table 2.5: Semi-supervised learning performance for graph classification in TUdatasets (Averaged accuracy $\pm$ std. over 10 runs). The **bold** numbers denote the best performance and the numbers in blue represent the second best performance

Dataset	NCI1	PROTEINS	DD	RDT-B	RDT-M5K	GITHUB	ogbg-molhiv	Avg.Rank
GAE	74.36 ± 0.24	70.51 ± 0.17	74.54 ± 0.68	87.69 ± 0.40	53.58 ± 0.13	63.89 ± 0.52	-	6.67
InfoGraph	74.86 ± 0.26	72.27 ± 0.40	75.78 ± 0.34	88.66 ± 0.95	53.61 ± 0.31	65.21 ± 0.88	-	4.50
GraphCL	74.63 ± 0.25	74.17 ± 0.34	76.17 ± 1.37	89.11 ± 0.19	52.55 ± 0.45	65.81 ± 0.79	55.48 ± 1.32	4.29
JOAO	74.48 ± 0.25	72.13 ± 0.92	75.69 ± 0.67	88.14 ± 0.25	52.83 ± 0.54	65.00 ± 0.30	56.83 ± 1.39	5.71
JOAOv2	74.86 ± 0.39	73.31 ± 0.48	75.81 ± 0.73	88.79 ± 0.65	52.71 ± 0.28	66.60 ± 0.60	57.39 ± 1.39	4.00
AD-GCL	75.18 ± 0.31	73.96 ± 0.47	$\mathbf{77.91 \pm 0.73}$	$\mathbf{90.10 \pm 0.15}$	53.49 ± 0.28	67.13 ± 0.52	-	2.33
AutoGCL	73.75 ± 2.25	$\mathbf{75.65 \pm 2.40}$	77.50 ± 4.41	79.80 ± 3.47	49.91 ± 2.70	62.46 ± 1.51	-	5.83
GPA	$\mathbf{75.50 \pm 0.14}$	74.27 ± 1.11	76.68 ± 0.81	$\mathbf{89.99 \pm 0.32}$	$\mathbf{54.92 \pm 0.35}$	$\mathbf{68.31 \pm 0.13}$	$\mathbf{60.76 \pm 1.01}$	1.57

2.3.2 Comparison with Baselines

We start by comparing the performance of GPA with the state-of-the-art baseline methods under two settings (**Q1**). Table 2.4 & Table 2.5 report the results of all methods on diverse datasets under unsupervised and semi-supervised settings, respectively. We have the following **Obs**ervations.

① **With personalized augmentations for each graph, GPA outperforms vanilla GCL methods with fixed augmentation per dataset.** By identifying different augmentations for different sample graphs, GPA performs generally better than vanilla GCL methods on two evaluation scenarios (Table 2.4 and Table 2.5). Specifically, in the semi-supervised evaluation task, GPA consistently outperforms GAE, InfoGraph, GraphCL, JOAO, and JOAOv2 on all datasets. In the unsupervised setting, GPA outperforms the vanilla GCL methods on 6 out of 8 datasets. In particular, GPA im-

proves 7.8%, 9.9%, 9.6%, and 6.7% over InfoGraph, JOAOv2, JOAO, and GraphCL on COLLAB in Table 2.4, respectively. This observation validates the effectiveness of performing personalized augmentation in GCL training.

② **Across diverse datasets, GPA performs better than (or on par with) the view-generated GCL methods on two evaluation settings.** On all datasets originating from diverse domains, GPA generally performs better or sometimes on par with the state-of-the-art AD-GCL and AutoGCL methods, as shown in the average rank. In unsupervised setting, GPA outperforms AD-GCL on all datasets while GPA beats AutoGCL on 6 out of 8 datasets. In semi-supervised setting, GPA outperforms both AD-GCL and AutoGCL on NCI1, RDT-B, RDT-M5K, and GITHUB datasets and achieves the second best on PROTEINS.

③ **GPA scales well on large datasets.** To study the scalability of our model, we further conduct experiment on the large-scale OGB dataset: ogbg-molhiv. View-generated GCL methods are excluded for this dataset since they are not officially tested on OGB datasets. From the Table 2.5, GPA consistently performs better than GraphCL and JOAO. To be specific, GPA improves 9.5% and 5.9% over GraphCL and JOAOv2 on ogbg-molhiv, respectively.

2.3.3 Personalized Augmentation Analysis

To study the effectiveness of our model in identifying informative augmentation types for various graphs (**Q2**), we visualize the learned augmentation distribution on Fig. 2.3. By comparing across different types of datasets, we observe the following.

④ **By learning from the data, GPA can effectively assign different augmentations for various datasets.** Our model GPA can identify different augmentations for different sample graphs, and allow different datasets to have their own augmentation distributions (see Fig. 2.3). Specifically, on MUTAG, 19 graphs prefer (*Identical, NodeDrop*) augmentations, while 30 graphs favor (*Subgraph, Subgraph*) augmentation combinations. Notice that (*Subgraph, Subgraph*) will generate two different subgraph-perturbation-induced augmented views, owning to sample randomness. Besides, COLLAB more likes the (*AttMask, AttMask*) augmentation pair, while DD prefers (*EdgePert, EdgePert*) operations. These observations empirically echo the necessity of performing

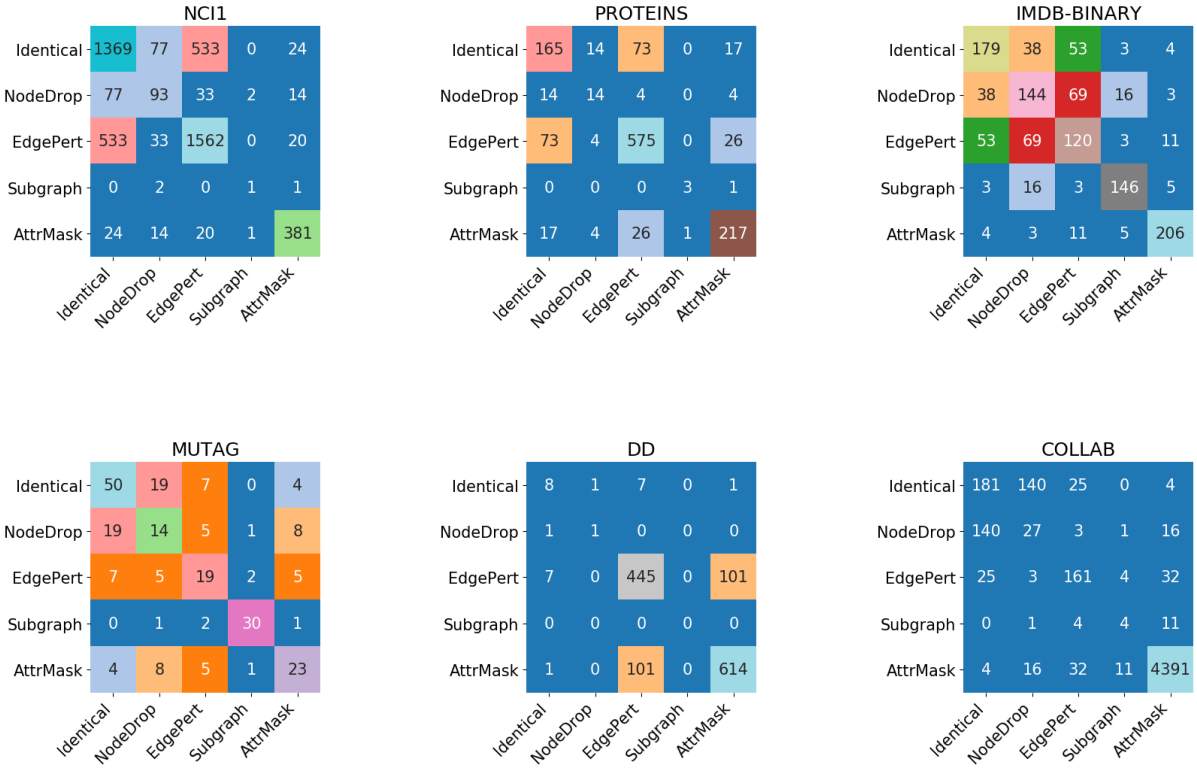

Figure 2.3: Augmentation distribution learned by GPA over molecules, bioinformatics, and social networks, in terms of the unsupervised setting.

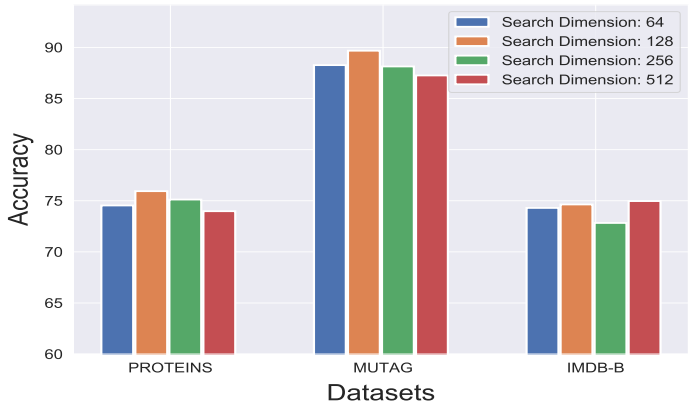

Figure 2.4: Personalized augmentation selector dimension analysis of GPA

Table 2.6: Ablation study of GPA under unsupervised setting in terms of mean classification accuracy

	GPA-random	GPA
NCI1	77.71 ± 0.60	$\mathbf{80.42 \pm 0.41}$
PROTEINS	74.21 ± 0.39	$\mathbf{75.94 \pm 0.25}$
DD	77.25 ± 0.69	$\mathbf{79.90 \pm 0.35}$
MUTAG	86.08 ± 2.93	$\mathbf{89.68 \pm 0.80}$
IMDB-B	71.80 ± 1.13	$\mathbf{74.64 \pm 0.35}$

personalized augmentation for GCL methods.

Another promising observation is that our model can assign (*Identical, Identical*) choice (i.e., two identical views) to some portion of graphs over all datasets. Given that the mutual information between two identical views (i.e., representations) is always maximized, such pure identical augmentations can be regarded as a skip operation. That is, these graphs abandon themselves during the GCL model training. The possible reason is that the existing augmentation strategies are not suitable to capture their characteristics or damage their semantic meanings. In this case, blindly selecting any combination of other augmentation types may incur huge performance degradation or noise. This observation sheds light on designing more advanced augmentation strategies beyond the current basic augmentations. On the other hand, it verifies the effectiveness of the proposed augmentation selector in skipping noisy graphs during model training by providing identical augmentations.

2.3.4 Ablation Study

To further investigate the effectiveness of the proposed personalized augmentation selector (**Q3**), we compare it with a random-search based variant, i.e., GPA-random. GPA-random replaces the personalized augmentation selector with a random mechanism. Specifically, it assigns one random augmentation pair to each graph. Noticed that GPA-random still assigns different augmentations to each graph while GraphCL assigns one pre-defined pair of augmentation strategies

to the whole dataset. Table 2.6 shows the results in terms of unsupervised setting. From the table, we can observe that GPA consistently performs better than GPA-random in all cases. In particular, GPA improves 3.5%, 2.3%, 3.4%, 4.2%, and 4.0% over GPA-random on NCI1, PROTEINS, DD, MUTAG, and IMDB-B, respectively. This comparison validates our motivation to develop a tailored and learnable personalized augmentation selector for GCL methods.

2.3.5 Parameter Sensitivity Analysis

We now study the impact of the parameter, i.e., the hidden dimension d of the score function to answer **Q4**. Specifically, we search d from the set $\{64, 128, 256, 512\}$ and plot the results of GPA on three representative datasets in Fig. 2.4. Similar observations are obtained by other datasets. From the figure, we can see that GPA generally performs stably across various dimension choices.

3. EFFICIENT GRAPH CONTRASTIVE TRAINING WITH MODEL PERTURBATION

To improve the training efficiency of GCL, in this work, we propose a novel contrastive learning framework, PerturbGCL, to train GNN encoders by customized model perturbations. Different from SimGRACE [2] that only focuses on weight perturbation, we further design effective model-level perturbation to provide local disturbances between contrastive views. Following prior work on GNN [109], we treat each layer of GNN as two main operations: message propagation (MP) and feature transformation, and then develop two tailored perturbation functions (*randMP* and *weightPrune*) to perturb them separately. Specifically, *randMP* offers local perturbation to the nodes in both views by performing a random number of message propagation steps at each GNN layer. This design is inspired by the fact that performing MP k times can be considered as a convolution [138] on the k-hops of neighbors of the anchor node. On this basis, we can learn diverged but correlated representations from two contrasting models with different k values due to the homophily theory [139]. *weightPrune* can provide weight-level perturbations by pruning the transformation weights of the target GNN branch on-the-fly. Unlike the Gaussian noise in [2], our pruned model will co-evolve with the target GNNs, leading to adaptively mining noise perturbations from the data, i.e., data-driven. Coupling these two strategies together yields a principled model perturbation solution tailored for the GCL.

3.1 Problem Statement

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X}, \mathbf{A})$ be an undirected graph, where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges. $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times D}$ is the node feature matrix where the i-th row of $\mathbf{X}$ denote the D-dimensional feature vector of the i-th node in $\mathcal{V}$. $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ indicates the adjacency matrix, where $\mathbf{A}_{i,j} = 1$ if the i-th node and j-th node is connected (i.e., $(i, j) \in \mathcal{E}$), otherwise $\mathbf{A}_{i,j} = 0$. We use f_w denote the mapping function that encodes each node $v \in \mathcal{G}$ into a d-dimensional representation $\mathbf{h}_v \in \mathbb{R}^d$.

Graph neural networks. To learn representations on graph data, we often utilize graph neural networks (GNN) [21, 22, 112] as the backbone encoder f_w. Without loss of generality, we present

GNN as a message passing network:

$$\mathbf{h}_v^{(l)} = \sigma(\mathbf{a}_v^{(l-1)}\mathbf{W}^l), \;\; \mathbf{a}_v^{(l-1)} = g^{(l-1)}(\mathbf{h}_v^{(l-1)}, \{\mathbf{h}_u^{(l-1)} : u \in \mathcal{N}_v\}), \tag{3.1}$$

where $\mathbf{h}_v^{(l)} \in \mathbb{R}^d$ is the intermediate representation of node v at the l-the layer, $\mathcal{N}_v$ denotes the direct neighbors of node v. We use g to denote the message propagation (MP) function, which updates node representations by integrating its neighbors using graph convolution, such as the spectral graph convolution in [21]. $\mathbf{W}^l \in \mathbb{R}^{d\times d}$ is the weight transformation matrix and σ is the activation function, such as ReLU.

Assume we have L layers in total; the final representation of node v is given by $\mathbf{h}_v = \mathbf{h}_v^L$, where $\mathbf{h}_v^L$ is the output of the L layer defined in Eq. (3.1). In addition to node representation, we are also interested in graph-level representation $\mathbf{h}_\mathcal{G} \in \mathbb{R}^D$ for graph-level applications. To this end, a readout function is often built upon the GNN layers to obtain the whole graph representation as:

$$\mathbf{h}_\mathcal{G} = \text{READOUT}(\{\mathbf{h}_v^L : v \in \mathcal{V}\}), \;\; \mathbf{z}_\mathcal{G} = h(\mathbf{h}_\mathcal{G}) = \text{MLP}(\mathbf{h}_\mathcal{G}). \tag{3.2}$$

Here READOUT$(\cdot)$ can be the simple average pooling function or more sophisticated ones [140]. $h(\cdot)$ is the projection head, and $\mathbf{z}_\mathcal{G} \in \mathbb{R}^D$ denotes the embedding towards loss estimation. In the development of our method, we follow the existing graph contrastive learning practices and consider three state-of-the-art GNNs: GCN [21], GIN [135], and ResGCN [136].

Problem formulation. Given the notations above, we formally define the problem of graph contrastive learning with model perturbation interested in this work as below.

Given an unlabeled graph $\mathcal{G}$ and a GNN encoder f_w, we aim to learn node (or graph) representations $\mathbf{H}$ (or $\mathbf{H}_\mathcal{G}$) using contrastive learning by creating contrastive samples via disturbing the GNN encoder f_w. As a result, not only is the training process more efficient compared to state-of-the-art data augmentation-based baseline methods, but also the learned representations can achieve satisfactory results in both node and graph classification tasks.

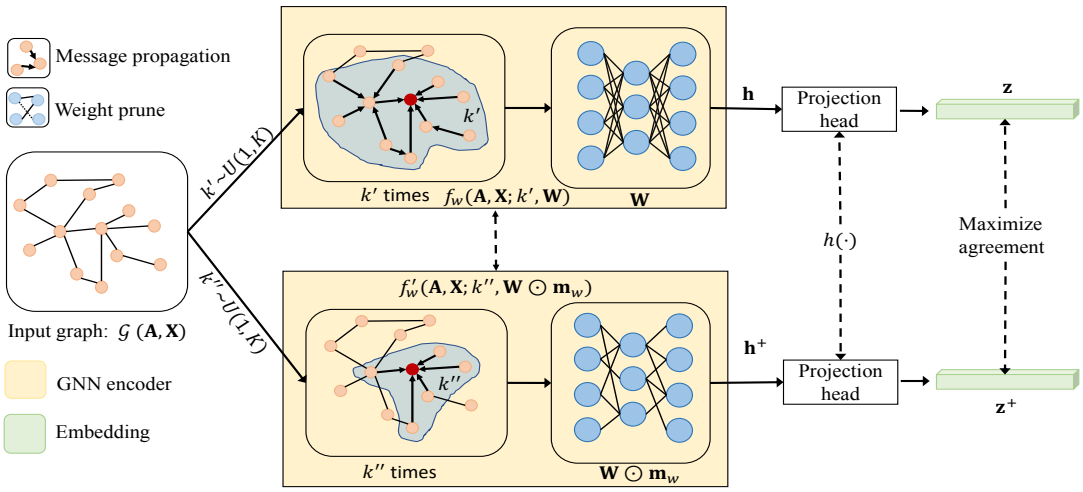

Figure 3.1: The overview of the proposed PerturbGCL framework. The original graph is fed into two asymmetric GNN branches: one is the target encoder f_w to be trained, and the other is the perturbed version f'_w that is pruned from the former online. The two branches share weights for their non-pruned parameters. Either branch has independent message propagation (MP) operations perturbed by a random number, i.e., k, to disturb nodes locally. Since the pruned branch is always obtained and updated from the latest target model, the two branches will co-evolve during training.

3.2 Methodology

To address the tradeoff between efficiency and effectiveness of existing GCL methods, we propose a principled approach to manipulate the structure of GNN models (see the overall paradigm in Figure 3.1) so that the models can be trained efficiently without degrading performance. In this section, we first elaborate the two customized perturbation functions in Section 3.2.1, and then show its relation to the data augmentation-based GCL approach in Section 3.2.2. Finally, we present how to empirically train our method in Section 3.2.3.

3.2.1 Perturbation Functions

As shown in Figure 3.1, the proposed framework follows an asymmetric architecture containing a target GNN branch f_w and an online encoder branch $f_w^{'}$, where f_w and $f_w^{'}$ share the same GNN architecture and weights. The main difference is that the degree of perturbations injected on the message propagation and model weights differ (we will illustrate this later). Given an input

graph $\mathcal{G}$, rather than performing data augmentations to generate two augmented views, we directly take the original graph as input and then output two different representations for the same node through two perturbed GNN encoders. After that, we add another projection head function $h(\cdot)$ to avoid representation collapse [141], which maps the latent space into the estimation space for loss computation. Note that we do not rely on tricks such as "stop gradient" to stabilize the training. Instead, the model parameters of two GNN branches will be jointly optimized via stochastic gradient descent, accelerating the model convergence.

Now, we will illustrate how to effectively disturb the two GNN branches in a principled manner. Motivated by the fact that GNN could be divided into the message passing and transformation operations [109], we design two types of model perturbations as below.

Perturbation function for message passing: *randMP*. Message passing [112] is the revolutionary design that generalizes convolution operation in regular data (e.g., images) to non-Euclidean graphs. Its success relies on determining a graph filter $\mathbf{F}$ to aggregate messages from neighboring nodes. Specifically, taking the spectral graph convolution in [21] for example, it adopts a fixed filter based on the graph characteristics (e.g., degree information) and formalize the message passing function $g(\cdot)$ as below.

$$\begin{aligned} &g(\mathbf{A}, \mathbf{X}) = \mathbf{F}\mathbf{X}, \\ &\text{where} \quad \mathbf{F} = \mathbf{D}^{-\frac{1}{2}}(\mathbf{A} + \mathbf{I}_{\mathcal{G}})\mathbf{D}^{-\frac{1}{2}}. \end{aligned} \tag{3.3}$$

$\mathbf{D} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ is a diagonal degree matrix where each entry on the diagonal is equal to the row-sum of the adjacency matrix, i.e., $\mathbf{D}_{i,i} = \sum_j \mathbf{A}_{i,j}$. $\mathbf{I}_{\mathcal{G}}$ denotes the identify matrix.

According to graph theory [54], $\mathbf{F}$ is related to the Laplacian matrix, which measures the degree to which a node differs from the values of nearby nodes. In other words, if we operate on the filter matrix $\mathbf{F}$, then it is similar to disturbing the original graph structure represented by $\mathbf{A}$, because the filter matrix can implicitly reflect the strength of the connection between two nodes in $\mathcal{G}$.

Inspired by this, we propose to disturb the filter matrix $\mathbf{F}$ to inject some noise for contrastive learning, namely *randMP*. Specifically, let k act as a perturbation factor which is randomly gen-

erated from a uniform distribution. The high-level idea of *randMP* is to randomly perform the convolution process k times in each GNN layer, instead of once. Formally, the perturbed version of Eq. (3.3) could be expressed as:

$$g(\mathbf{A}, \mathbf{X}) = \mathbf{F}^k\mathbf{X} = \underbrace{\mathbf{F}...\mathbf{F}}_{k-1}\mathbf{F}\mathbf{X}, \tag{3.4}$$

where $\mathbf{F}^k$ is the matrix power. In the case of the GNN domain, Eq. (3.4) is equivalent to updating the node representation according to its k-hop local subgraph. Note that we do not need to compute the expensive $\mathbf{F}^k$ term in practice, since it can be approximated by repeating the convolution process ($\mathbf{FX}$) k times. This perturbation is also similar to the higher-order convolution trick in [142], but we apply it to model perturbations for contrastive learning.

Benefits. By perturbing the message propagation using a random factor k, it is possible to include more high-order neighbors for GNN encoding, which enriches the training set from the perspective of local subgraphs. On the other hand, this perturbation is much safer for contrastive learning than standard data augmentation, such as edge deletion and addition, since k controls the orders of neighbors of the anchor node. That is, different values of k for two GNN branches indicate that we utilize neighbors with different hops to obtain node representation, which are both still centered on the anchor node. Therefore, their outputs should be similar to some extent according to the homophily theory.

Perturbation function for feature transformation: *weightPrune*. In addition to graph convolution, feature transformation is another critical component of each GNN layer as it contains the primary trainable parameters $\mathbf{W}$ operating on the feature space, simplified as:

$$\mathbf{H} = \mathbf{FXW}. \tag{3.5}$$

In fact, the training goal of graph contrastive learning is to learn the set of weight matrices $\{\mathbf{W}^l\}_{l=1}^L$ from the GNN encoder with L layers. Recently, work in [61] found that GNN encoder maybe over-parameterized, since a sparse sub-network can be identified to achieve on par results with

the complete one under the lottery ticket hypothesis [143]. Inspired by this, we argue that the GNN encoders in contrastive learning could be also over-parameterized and propose to enforce the sparsity of the learned GNN encoder to improve the generalization ability. In particular, we develop *weightPrune*, which creates the perturbed GNN branch by pruning the model parameters of the target GNN encoder. Formally, assume $s \in [0, 1]$ is the pre-defined prune ratio, $\mathbf{M}_w \in \{0, 1\}$ indicate the mask indicator of weight matrix $\mathbf{W} \in \mathbb{R}^{d\times d}$, we perturb the weight matrix of the online encoder as follows.

$$\begin{aligned} \mathbf{H} &= \mathbf{F}\mathbf{X}(\mathbf{W} \odot \mathbf{M}_w). \\ \text{s.t.} \quad ||\mathbf{M}_w||_0 &= (1-s)d^2, \end{aligned} \tag{3.6}$$

where $||\cdot||_0$ is the l_0-norm and $||\mathbf{M}_w||_0$ indicates the number of nonzero elements in $\mathbf{M}_w$. By changing s, we can control the distortion degree of the target GNN branch to a certain extent, i.e., sd^2 elements in the $\mathbf{W}$ is reserved. Note that different from [61] which aims to identify the optimal sub-network for model pruning, we adopt model pruning as a perturbation function to disturb the weight matrix of target GNN encoder in contrastive learning to generate contrastive views.

Benefits. By perturbing the model weights using pruning technique with ratio s, the mask indicator $\mathbf{M}_w$ will be continuously updated from the latest target model, making the two GNN branches in Figure 3.1 co-evolve during training. In addition, unlike simply injecting Guassian noise to model weights as done in [2], the perturbation pattern in our case is dynamic and data-driven, since the mask indicator $\mathbf{M}_w$ can change along the optimization. This dynamic nature makes our *weightPrune* works well over both node and graph application scenarios.

In summary, we propose a principled approach to effectively perturb GNN architectures from the message passing and feature transformation aspects through two easily implemented perturbation functions: *randMP* and *weightPrune*. *randMP* aims to map the same input graph into two semantically similar representations by performing a random number of message-passing steps. Meanwhile, *weightPrune* seeks to increase the diversity of the two representations by dynamic pruning. Combining these two strategies, we are able to find the sweet spot [144] between two view representations that are related but sufficiently divergent.

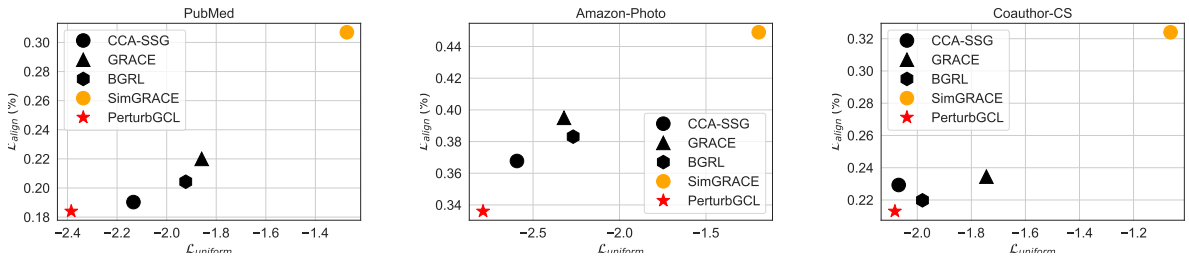

Figure 3.2: The *alignment* and *uniformity* plot for BGRL [1], SimGRACE [2], GRACE [3], CCA-SSG [4], and our PerturbGCL on the same perturbed graphs generated by data augmentation. *Black circles* indicate the baselines. *Orange circles* represent the performance of SimGRACE. *Red starts* are the results of PerturGCL.

3.2.2 Why is PerturbGCL Effective?

Data augmentation [145] has become the common practice for generating augmented views in graph contrastive learning. Although several augmentation strategies have been proposed recently, the key insight is how to effectively disturb the structure information in $\mathbf{A}$ and feature information in $\mathbf{X}$. For example, Zhu *et al.* [146] empirically observed that the most successful augmentation choice for node representation learning is a combined scheme consisting of structure- and feature-level perturbations. You *et al.* [48] also found that composing augmentations from the structure and feature aspects are beneficial, since they provide relatively difficult and comprehensive perturbations on the graph. Formally, let $\mathbf{H}^{DA}$ indicate the embedding matrix of data augmentation based method, and $T_1 \in \mathcal{T}$ and $T_2 \in \mathcal{T}$ denote the structure- and feature-level augmentation functions, where $\mathcal{T}$ is the augmentation distribution. Following [109], the forward propagation of GNN encoder in GCL could be simplified as:

$$\mathbf{H}^{DA} = \mathbf{F}^{T_1}\mathbf{X}^{T_2}\mathbf{W}. \tag{3.7}$$

$\mathbf{F}^{T_1} = \hat{\mathbf{D}}^{-\frac{1}{2}}(\hat{\mathbf{A}} + \mathbf{I}_{\mathcal{G}})\hat{\mathbf{D}}^{-\frac{1}{2}}$ represents the augmentation-based graph filter, wherein $\hat{\mathbf{A}}$ is the perturbed adjacency matrix after applying augmentation function T_1 on $\mathbf{A}$, such as edge perturbation (e.g., edge deletion or addition), and $\hat{\mathbf{D}}_{ii} = \sum_j \hat{\mathbf{A}}_{i,j}$. $\mathbf{X}^{T_2}$ is the perturbed feature matrix after

applying augmentation function T_2 on $\mathbf{X}$, such as attribute masking.

Connection to data augmentation. We analyze the theoretical connection between the proposed PerturbGCL and the popular graph augmentation-based approach to explain why PerturbGCL can learn comparable or even more informative representations. In a nutshell, we find that our PerturbGCL actually creates view representations similar to standard data augmentation, but in a global way, thanks to the design of perturbation functions for message passing (i.e., *randMP*) and transformation (i.e., *weightPune*).

Formally, let $g_k(\mathbf{A}; k)$ and $p(\mathbf{W}; s)$ denote the perturbation functions on GNN's message passing and transformation operations, as defined in Eq. (3.4) and Eq. (3.6) respectively. The forward propagation of the GNN encoder in our PerturbGCL can be simplified as:

$$\begin{aligned} \mathbf{H} =& g_k(\mathbf{A}; k)p(\mathbf{W}; s) = \underbrace{\mathbf{F}...\mathbf{F}}_{k}\mathbf{X}\mathbf{W} \odot \mathbf{M}_w. \\ &= (\mathbf{F}^k)(\mathbf{X}\mathbf{W} \odot \mathbf{M}_w). \end{aligned} \tag{3.8}$$

Here, we misuse the notation of g_k and p depending on the adjacency matrix $\mathbf{A}$ and feature matrix $\mathbf{X}$, respectively. We can find that Eq. (3.8) is a special implementation of Eq. (3.7) by: 1) designing the local edge perturbation as $g_k(\mathbf{A}; k)$, which acts globally on the graph filter matrix $\mathbf{F}$ instead of the original adjacency matrix $\mathbf{A}$; and 2) treating the local attribute masking on the feature dimension as a global perturbation function $p(\mathbf{W}; s)$ of the latent space. Note that $p(\mathbf{W}; s)$ can mask some feature dimensions by specifying the corresponding columns in $\mathbf{M}_w$ as zeros. In conclusion, the correspondence between the data augmentation strategy and our PerturbGCL proposal in terms of structure-level and feature-level interference guarantees its effectiveness. Meanwhile, unlike data augmentation that requires disturbing the original graph locally, the global operational nature of our PerturbGCL from the GNN encoder perspective sheds light on its efficiency in training.

3.2.3 Model Optimization

Given a graph $\mathbf{G}$ as input, the two GNN branches in Figure 3.1 will output two representations for the same node (or graph). Specifically, let $(\mathbf{z}_v, \mathbf{z}_v^+)$ denote the representations of positive pairs

of node v, and $(\mathbf{z}_v, \mathbf{z}_v^-)$ be the negative pairs. We follow [48] about the representations of different nodes as negative samples. Then, our model can be trained to minimize the standard instance-wise InforNCE objective, expressed as:

$$\mathcal{L}_v = -\log \frac{\exp(\text{sim}(\mathbf{z}_v, \mathbf{z}_v^+)/\tau)}{\exp(\text{sim}(\mathbf{z}_v, \mathbf{z}_v^+)/\tau) + \sum_{u \in \mathcal{V}, u \neq v} \exp(\text{sim}(\mathbf{z}_v, \mathbf{z}_u^-)/\tau)}, \tag{3.9}$$

where τ is the temperature parameter, $\text{sim}(\cdot, \cdot)$ denotes the similarity function. By minimizing Eq. (3.9), the GNN encoder will be trained to enforce the similarity of the positive pairs while enlarging the distance of negative pairs in the hidden space. It is also worth noting that some GCL variants [1, 147] do not rely on negative samples, such as Barlow Twins [98] and Bootstrap [1].

3.2.4 Sanity Check

In this section, we analyze the effectiveness of representations learned by our PerturbGCL. To benchmark the quality of representations, we consider two popular metrics: *alignment* and *uniformity* introduced in [148], expressed as:

$$\begin{aligned} \mathcal{L}_{\text{align}}(f_w; \alpha) &\triangleq \mathbb{E}_{(x,y)\sim P_{\text{pos}}}[||\mathbf{h}_x - \mathbf{h}_y||_2^{\alpha}],\ \alpha > 0 \\ \mathcal{L}_{\text{uniform}}(f_w; t) &\triangleq \log \mathbb{E}_{(x,y) \overset{\text{i.i.d}}{\sim} P_{\text{data}}}[e^{-t||\mathbf{h}_x - \mathbf{h}_y||_2^2}],\ t > 0. \end{aligned} \tag{3.10}$$

P_{pos} is the distribution of positive pairs, i.e., augmentations of the same sample, P_{data} is the data distribution. $\mathcal{L}_{\text{align}}$ is used to measure if positive samples stay close in the hidden space. $\mathcal{L}_{\text{uniform}}$ is used to measure if random samples are scattered on the hypersphere of hidden space. In our experiments, we set $\alpha = t = 2$ following [2]. For all methods, we first pre-train them according to their official configurations. Then, we follow BGRL [1] to create the augmented views using data augmentation to compute $\mathcal{L}_{align}$ and $\mathcal{L}_{uniform}$.

Figure 3.2 reports the test results of several state-of-the-art data augmentation-based GCL methods [4, 1, 3] and the naive model perturbation method (SimGRACE [2]) on the same augmented graphs. As can be observed, the proposed PerturbGCL consistently outperforms strong data augmentation baselines across three datasets. However, SimGRACE performs substantially

worse than these data augmentation baselines. This is mainly because SimGRACE only focuses on weight perturbation and does not support message passing-level perturbation. These results shed light on the effectiveness of PerturbGCL in learning informative representations compared with data augmentation competitors.

Algorithm 2 The pre-training procedure of PerturbGCL

Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, GNN encoder $f_w(\cdot)$ with weight $\mathbf{W}$, projection head $h(\cdot)$, the maximum propagation step K, and the pruning ratio s
Initialize the encoder $f_w(\cdot)$ and head $h(\cdot)$.
Set mask indicator matrix $\mathbf{M}_w$ to ones.
for iterate 1, 2, ... times until convergence **do**
    Sample the random propagation steps k', k'' from the uniform distribution $\sim U(1, K)$.
    Conduct weight pruning to update the mask indicator $\mathbf{M}_w$
    Compute the target representation $\mathbf{h}_v$ based on f_w and the message passing perturbation function $g(\mathbf{A}; k')$.
    Compute the online representation $\mathbf{h}_v^+$ based on f_w, mask indicator $\mathbf{M}_w$, and the message passing perturbation function $g(\mathbf{A}; k'')$.
    Optimize model weights according to Eq. (3.9).
end for
return the pre-trained GNN encoder $f_w(\cdot)$

3.2.5 Complexity Analysis

In this section, we show why PerturbGCL is efficient during training. In addition to the significant time savings in searching for optimal graph enhancement strategies, we analyze the complexity of PerturbGCL. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and the GNN encoder f_w. The time complexity for most popular GNN architectures [21, 23, 112] is $\mathcal{O}(|\mathcal{E}| + |\mathcal{V}|)$, where $\mathcal{O}(|\mathcal{E}|)$ and $\mathcal{O}(|\mathcal{V}|)$ are mainly caused by the message propagation and feature transformation operations, respectively. PerturbGCL performs two encoder computations per update step (one for each GNN branch) plus a node-level projection head. Assuming that the backward pass is approximately as costly as a forward pass and ignoring the cost for weight pruning as it is small and negligible. Thus the total time complexity per update step for PerturbGCL is $4C_{\text{encoder}}(K|\mathcal{E}| + |\mathcal{V}|s) + 2C_{\text{head}}(|\mathcal{V}|) + C_{\text{loss}}$, where

Algorithm 3 PerturbGCL on graph level task

Input: A set of graphs $\mathcal{G} = \{\mathcal{G}_n\}_{n=1}^{N}$, where $\mathcal{G}_n = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, GNN encoder $f_w(\cdot)$ with weight $\mathbf{W}$, projection head $h(\cdot)$, the maximum propagation step K, and pruning ratio s
Initialize the encoder $f_w(\cdot)$.
Initialize the mask indicator matrix $\mathbf{M}_w$ to ones.
for iterate 1, 2, ... times until convergence **do**
  for sampling batches $\mathcal{G}_n$ from $\mathcal{G}$, where $n = 1, 2, ..., N$ **do**
    Sample the random propagation steps k', k'' from the uniform distribution $\sim U(1, K)$.
    Conduct weight pruning to update the mask indicator $\mathbf{M}_w$.
    Compute the target representation $\mathbf{h}_v$ based on f_w and the message passing perturbation function $g(\mathbf{A}; k')$.
    Compute the online representation $\mathbf{h}_v^+$ based on f_w, mask indicator $\mathbf{M}_w$, and the message passing perturbation function $g(\mathbf{A}; k'')$.
    Obtain the graph-level representation $\mathbf{h}_{\mathcal{G}_n}$ and $\mathbf{h}_{\mathcal{G}_n}^+$ via readout function.
    Optimize model weights according to Eq. (3.9).
  end for
end for
return The pre-trained GNN encoder $f_w(\cdot)$

$C_\cdot$ are constants depending on the architecture of the different components, K is the maximum number of MP operations considered (e.g., $K = 3$), and s is the pruning ratio.

It is worth noting that although our model at most takes K times MP operations in the forward pass, due to weight pruning (e.g., $s = 70\%$), the computation costs for feature transformation and backpropagation are significantly lower (i.e., 30% of weights) than standard GCL methods. Therefore, the overall running cost of PerturbGCL per epoch can be further accelerated in practice. More importantly, we empirically observe that our PerturbGCL model can significantly speed up the convergence (i.e., the total number of training epochs) as shown in Section 3.3.5, due to the online perturbation fashion. Algorithm 2 and 3 summarizes the optimization procedure of PerturbGCL on node and graph classification tasks, respectively.

3.3 Experiments

In this section, we try to answer the following research questions through experiments.

- **RQ1** What the proposed two strategies actually do?

- **RQ2** How effective is PerturbGCL compared with state-of-the-art contrastive learning baselines over two classification scenarios?
- **RQ3** How efficient is PerturbGCL compared with strong baseline methods?
- **RQ4** What are the contributions of different components in PerturbGCL?
- **RQ5** What is the impact of hyper-parameters, such as propagation factor and prune ratio, on PerturbGCL?

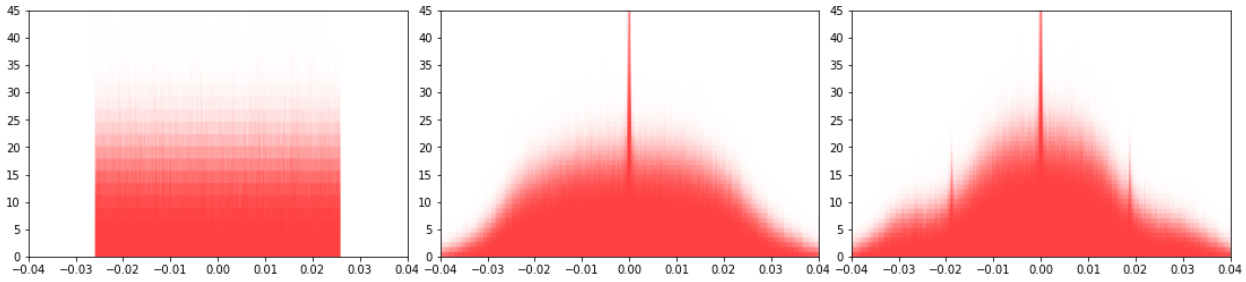

Figure 3.3: Visualization of weight distribution (from left to right: initial weights, PerturbGCL w/o. *weightPrune*, and PerturbGCL) on Coauthor-Phy. The x-axis indicates weight values and y-axis is the count. Obviously, the number of activated neurons after using *weightPrune* is significantly smaller than others. It shows that *weightPrune* can regularize the model.

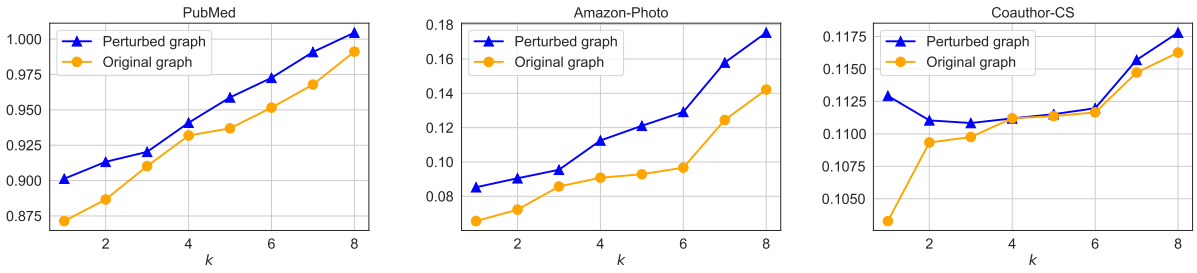

Figure 3.4: The visualization of PerturbGCL w.r.t. different k values on the original graphs and the perturbed graphs generated by data augmentation. The x-axis indicates propagation steps and y-axis is the $\mathcal{L}_{\text{align}} \downarrow$. The gap between the blue and orange lines indicate the generalization ability. Apparently, performing more MP steps will increase the diversity of two positive views since $\mathcal{L}_{\text{align}}$ increases. Sweet spots (i.e., minimum performance gap) exist across three scenarios.

3.3.1 Datasets and Experimental Settings

Datasets. For a comprehensive comparison, we use fourteen benchmark datasets from two crucial graph analysis scenarios: node classification and graph classification. For node classification task, we consider six popular datasets including two Planetoid datasets (**Cora** and **PubMed** [149]), and four popular contrastive learning benchmarks (including two co-purchase networks from Amazon [150]: Amazon-Computer **Compute**), Amazon-Photo (**Photo**), and two academic networks from Microsoft Academic Graph [151]: Coauthor-CS (**CS**), and Coauthor-Physics (**Phy**). For graph classification task, we consider eight graph-level datasets, including five social networks (**COLLAB**, REDDIT-BINARY (**RDT-B**), REDDIT-MULTI-5K (**RDT-M5K**), IMDB-BINARY (**IMDB-B**), and **GITHUB**)), and two molecules networks (**NCI1** and **MUTAG**), and two bioinformatics networks (**PROTEINS** and **DD**) from the benchmark TUDdataset [152]. We summarize their statistics in Tables 5.2 and 5.3.

Competitors. To demonstrate the effectiveness, we compare our model with state-of-the-art GCL methods of two fields. For node classification task, we consider eight data augmentation-based GCL methods (DGI [51], MVGRL [50], GRACE [3], GCA [52], BGRL [1], InfoGCL [99], CCA-SSG) [4], three augmentation-free GCL methods (AFGRL [59], SimGRACE [2], and MA-GCL [153]), and two supervised GNNs (GCN [21] and GAT [23]). For graph classification task, we consider three kernel-based methods ((GL) [154], Weisfeiler-Lehman sub-tree kernel (WL) [155], and deep graph kernel (DGK) [156]), three network embedding methods (node2vec [102], sub2vec [157], and graph2vec [158]), four GCL methods with data augmentation (MVGRL [50], InfoGraph [99], GraphCL [48], and JOAO [53]), and two GCL methods without data augmentation (SimGRACE [2], and MA-GCL [153])

Implementation details. For a fair comparison, we follow previous protocols [1, 48] to set up the node-level and graph-level evaluation scenarios. For baseline methods, we report the results published in original papers when available. We implement PerturbGCL with PyTorch and use Adam optimizer to train the model. The graph encoder f_w is specified as a standard two-layer GCN model for all the datasets. We have two hyperparameters (pruning ratio s and random

Table 3.1: Test accuracy on benchmark datasets in TUdatasets in terms of the unsupervised setting for graph classification. − means that results are not available in published papers. A.R. denotes the averaged rank.

Methods	NCI1	PROTEINS	DD	MUTAG	COLLAB	RDT-B	RDT-M5K	IMDB-B	A.R. ↓
GL	–	–	–	81.66 ± 2.11	–	77.34 ± 0.18	41.01 ± 0.17	65.87 ± 0.98	10.50
WL	$\mathbf{80.01 \pm 0.50}$	72.92 ± 0.56	–	80.72 ± 3.00	–	68.82 ± 0.41	46.06 ± 0.21	72.30 ± 3.44	8.17
DGK	$\mathbf{80.31 \pm 0.46}$	73.30 ± 0.82	–	87.44 ± 2.72	–	78.04 ± 0.39	41.27 ± 0.18	66.96 ± 0.56	7.33
node2vec	54.89 ± 1.61	57.49 ± 3.57	–	72.63 ± 10.20	–	–	–	–	11.67
sub2vec	52.84 ± 1.47	53.03 ± 5.55	–	61.05 ± 15.80	–	71.48 ± 0.41	36.68 ± 0.42	55.26 ± 1.54	12.50
graph2vec	73.22 ± 1.81	73.30 ± 2.05	–	83.15 ± 9.25	–	75.78 ± 1.03	47.86 ± 0.26	71.10 ± 0.54	9.00
MVGRL	–	–	–	75.40 ± 7.80	–	82.00 ± 1.10	–	63.60 ± 4.20	10.67
InfoGraph	76.20 ± 1.06	74.44 ± 0.31	72.85 ± 1.78	89.01 ± 1.13	70.65 ± 1.13	89.53 ± 0.84	$\mathbf{55.99 \pm 0.28}$	73.03 ± 0.87	4.13
GraphCL	77.87 ± 0.41	74.39 ± 0.45	$\mathbf{78.62 \pm 0.40}$	86.80 ± 1.34	71.36 ± 1.15	89.53 ± 0.84	$\mathbf{55.99 \pm 0.28}$	71.14 ± 0.44	4.50
JOAO	78.07 ± 0.47	74.55 ± 0.41	77.32 ± 0.54	87.35 ± 1.02	69.50 ± 0.36	85.29 ± 1.35	55.74 ± 0.63	70.21 ± 3.08	6.50
JOAOv2	78.36 ± 0.53	74.07 ± 1.10	77.40 ± 1.15	87.67 ± 0.79	69.33 ± 0.34	86.42 ± 1.45	$\mathbf{56.03 \pm 0.27}$	70.83 ± 0.25	5.63
SimGRACE	79.12 ± 0.44	75.35 ± 0.09	77.44 ± 1.11	89.01 ± 1.31	71.72 ± 0.82	$\mathbf{89.51 \pm 0.89}$	$\mathbf{55.91 \pm 0.34}$	71.30 ± 0.77	3.50
MA-GCL	78.76 ± 0.34	75.26 ± 0.17	77.47 ± 0.82	88.38 ± 1.04	72.35 ± 0.66	87.52 ± 0.67	55.71 ± 0.52	71.55 ± 0.84	4.13
PerturbGCL	$\mathbf{80.24 \pm 0.45}$	$\mathbf{76.08 \pm 0.30}$	$\mathbf{78.33 \pm 0.37}$	$\mathbf{89.97 \pm 0.50}$	$\mathbf{75.06 \pm 0.87}$	88.98 ± 0.67	$\mathbf{55.78 \pm 0.72}$	$\mathbf{74.14 \pm 0.50}$	2.13

propagation step K) to tune. For each dataset, we search $K \in [1, 2, 3]$ and $s \in [0.5, 0.7, 0.9]$. To avoid randomness, we report the mean accuracy with a standard deviation through 10 random initialization.

Table 3.2: Test accuracy on benchmark datasets in TUdatasets in terms of semi-supervised graph classification.

Dataset	NCI1	PROTEINS	DD	COLLAB	RDT-B	RDT-M5K	GITHUB	A.R. ↓
No pre-train	73.72 ± 0.24	70.40 ± 1.54	73.56 ± 0.41	73.71 ± 0.27	86.63 ± 0.27	51.33 ± 0.44	60.87 ± 0.17	9.86
Augmentations	73.59 ± 0.32	70.29 ± 0.64	74.30 ± 0.81	74.19 ± 0.13	87.74 ± 0.39	52.01 ± 0.20	60.91 ± 0.32	9.00
GAE	74.36 ± 0.24	70.51 ± 0.17	74.54 ± 0.68	75.09 ± 0.19	87.69 ± 0.40	53.58 ± 0.13	63.89 ± 0.52	7.00
InfoGraph	74.86 ± 0.26	72.27 ± 0.40	75.78 ± 0.34	73.76 ± 0.29	88.66 ± 0.95	53.61 ± 0.31	65.21 ± 0.88	5.43
ContextPred	73.00 ± 0.30	70.23 ± 0.63	74.66 ± 0.51	73.60 ± 0.37	84.76 ± 0.52	51.23 ± 0.84	–	10.50
GraphCL	74.63 ± 0.25	$\mathbf{74.17 \pm 0.34}$	$\mathbf{76.17 \pm 1.37}$	74.23 ± 0.21	$\mathbf{89.11 \pm 0.19}$	52.55 ± 0.45	65.81 ± 0.79	3.86
JOAO	74.48 ± 0.25	72.13 ± 0.92	75.69 ± 0.67	$\mathbf{75.30 \pm 0.32}$	88.14 ± 0.25	52.83 ± 0.54	65.00 ± 0.30	6.00
JOAOv2	74.86 ± 0.39	73.31 ± 0.48	75.81 ± 0.73	$\mathbf{75.53 \pm 0.18}$	88.79 ± 0.65	52.71 ± 0.28	66.60 ± 0.60	3.71
SimGRACE	74.60 ± 0.41	$\mathbf{74.03 \pm 0.51}$	$\mathbf{76.48 \pm 0.52}$	74.74 ± 0.28	88.86 ± 0.62	53.97 ± 0.64	65.33 ± 0.35	3.43
MA-GCL	74.35 ± 0.21	73.46 ± 0.44	76.02 ± 0.37	73.87 ± 0.35	88.57 ± 0.47	54.14 ± 0.38	66.40 ± 0.49	5.00
PerturbGCL	$\mathbf{75.23 \pm 0.52}$	$\mathbf{74.11 \pm 0.42}$	$\mathbf{76.65 \pm 0.57}$	74.50 ± 0.41	88.69 ± 0.63	$\mathbf{55.39 \pm 0.12}$	$\mathbf{68.40 \pm 0.10}$	2.14

3.3.2 What are *weightPrune* and *randMP* Doing? A Case Study

To answer **RQ1**, we visualize the weight distribution of PeruturbGCL during training in Figure 3.3. It shows that ① **by continuously pruning the target model along the training, *weight-***

***Prune* can regularize the target model progressively.** From the weight histograms in Figure 3.3, we can see that the bars around zero become higher and higher, which indicates more neurons are inactivated in the end. This observation shows the regularization effect of *weightPrune*. Since effective regularization can improve the generalization ability of neural networks, we believe that why the proposed *weightPrune* can improve the performance.

To investigate the effect of *randMP*, we report the impacts of different k values on PerturbGCL in terms of the alignment between positive views. From Figure 3.4, we observe that ② ***randMP* can improve the diversity of contrastive views when k increases, and sweet points widely exist across three datasets.** In Figure 3.4, with the increase of k, the generalization gap tends to first decrease to the sweet points and then increase a little bit. It validates the effect of *randMP* in generating correlated but diverged views. On the other hand, it indicates the potential of *randMP* to improve the generalization ability, i.e., these sweet points.

3.3.3 Comparison on Graph Classification Task

To validate the effectiveness of PerturbGCL on graph classification (**RQ2**), we compare it with strong graph-level GCL methods on different datasets. Table 3.1 and Table 3.2 report the results on unsupervised and semi-supervised settings, respectively. We made one major observation.

③ **PerturbGCL generally outperformed the augmented and free GCL-based baselines in both graph learning scenarios**. In the unsupervised setting (see Table 3.1), PerturbGCL achieves the best (or fairly good) results on 6 of the 8 datasets and substantial improvements on the COLLAB and IMDB-B datasets. In the semi-supervised setting (see Table 3.2, PerturbGCL generally outperformed the other baselines in seven comparisons, consistently ranking in the top three on all datasets. These results demonstrate the effectiveness of PerturbGCL on graph learning tasks. Another interesting observation is that MA-GCL always performs worse than PerturbGCL in both evaluation cases. We believe this is because GNN models tend to overfit graph-level datasets, which are small in size compared to node-level graphs. Therefore, perturbations to the model weights, as done in PerturbGCL, generalize better than in MA-GCL, which does not consider perturbations at the model weights.

Table 3.3: Test accuracy on benchmark datasets in terms of node classification. We report both mean accuracy and standard deviation. A.R. denotes the averaged rank.

	Method	Cora	PubMed	Compute	Photo	CS	Phy	A.R. ↓
Supervised	GCN	81.5	79.0	86.51 ± 0.54	92.42 ± 0.22	93.03 ± 0.31	95.65 ± 0.16	9.17
	GAT	83.0 ± 0.7	79.0 ± 0.3	86.93 ± 0.29	92.56 ± 0.35	92.31 ± 0.24	95.47 ± 0.15	8.83
Data augmentation	DGI	82.3 ± 0.6	76.8 ± 0.6	83.95 ± 0.47	91.61 ± 0.22	92.15 ± 0.63	94.51 ± 0.52	11.50
	MVGRL	83.5 ± 0.4	80.1 ± 0.7	87.52 ± 0.11	91.74 ± 0.07	92.11 ± 0.12	95.33 ± 0.03	8.33
	GRACE	81.9 ± 0.4	80.6 ± 0.4	$\mathbf{89.53 \pm 0.35}$	92.78 ± 0.45	91.12 ± 0.20	–	7.20
	GCA	83.4 ± 0.3	80.3 ± 0.4	87.85 ± 0.31	92.49 ± 0.09	93.10 ± 0.01	95.68 ± 0.05	6.00
	BGRL	81.8 ± 0.3	80.4 ± 0.5	$\mathbf{89.68 \pm 0.31}$	92.87 ± 0.27	93.21 ± 0.18	95.56 ± 0.12	5.50
	InfoGCL	83.5 ± 0.3	79.1 ± 0.2	–	–	–	–	6.00
	GBT	–	–	88.14 ± 0.33	92.63 ± 0.44	92.95 ± 0.17	95.07 ± 0.17	8.00
	CCA-SSG	$\mathbf{84.2 \pm 0.4}$	81.6 ± 0.4	88.74 ± 0.28	93.14 ± 0.14	93.31 ± 0.22	95.38 ± 0.06	3.83
Augmentation-free	AFGRL	82.3 ± 0.4	79.7 ± 0.2	$\mathbf{89.88 \pm 0.33}$	93.22 ± 0.28	93.27 ± 0.17	95.69 ± 0.10	4.33
	SimGRACE	78.5 ± 0.3	79.3 ± 0.5	86.42 ± 0.35	91.55 ± 0.22	92.37 ± 0.33	94.37 ± 0.15	11.33
	MA-GCL	83.3 ± 0.4	$\mathbf{83.5 \pm 0.4}$	88.83 ± 0.30	$\mathbf{93.80 \pm 0.10}$	94.19 ± 0.10	95.12 ± 0.22	3.67
	PerturbGCL	83.3 ± 0.5	82.10 ± 0.3	88.45 ± 0.77	$\mathbf{93.62 \pm 0.40}$	$\mathbf{94.58 \pm 0.09}$	$\mathbf{95.85 \pm 0.08}$	3.00

3.3.4 Comparison on Node Classification Task

To answer **RQ2**, we first examine the effectiveness of PerturbGCL in node classification. Results of 13 baseline methods across 6 benchmark datasets are collected in Table 3.3. We make the following observations.

④ **PerturbGCL usually performs better than SOTA GCL methods using data augmentation.** From table 3.3, PerturbGCL obtained the four best results in six evaluation scenarios compared to strong GCL methods such as BGRL and CCA-SSG. These results validate the effectiveness of training contrast learning models on graphs without explicit data augmentation.

⑤ **PerturbGCL achieved very competitive results compared to three augmentation-free GCL baselines**. In six datasets, PerturbGCL outperformed AFGRL and SimGRACE in most cases. Specifically, PerturbGCL improved over SimGRACE on Cora, PubMed, Computer, Photo, CS, and Phy by 6.11%, 3.53%, 2.34%, 2.26%, 1.95%, and 1.56%, respectively. MA-GCL was a contemporary work, and PerturbGCL performed slightly better than MA-GCL in general. Specifically, PerturbGCL won MA-GCL on CS and Phy, lost to PubMed, and tied with Cora, Compute, and Photo datasets. However, since CS and Phy are much larger datasets than the others, our method may scale better on practical large-scale graphs.

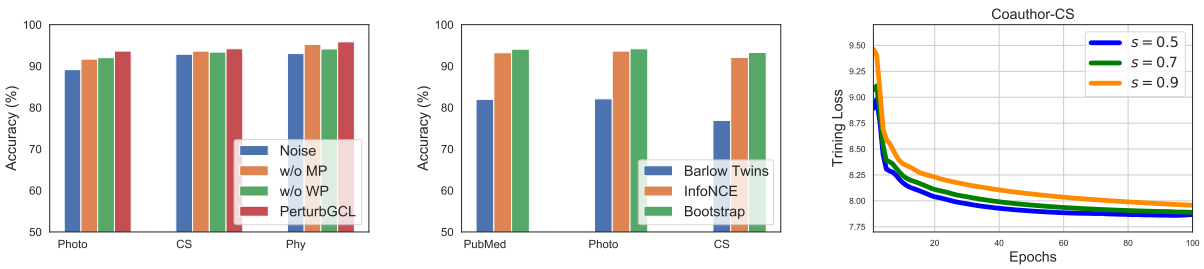

Figure 3.5: **Left:** Ablation study of PerturbGCL. **Middle:** The impact of different contrastive objectives. **Right:** Empirical training curves of PerturbGCL with different s values.

Table 3.4: Running time per epoch (in seconds). Baseline indicates BGRL and GraphCL for node and graph classification, respectively. All the methods are evaluated on GeForce RTX 2080 Ti GPUs. "method"-(x)-(y) implies that this method requires x and y epochs of training to converge on node and graph benchmarks, respectively. Considering running time per epoch and the total training epochs, PerturbGCL runs faster than all methods.

	Node Benchmark				Graph Benchmark			
	PubMed	Computer	Photo	CS	NCI1	COLLAB	RDT-B	RDT-M2K
Baseline-(1000)-(20)	0.14	0.16	0.08	0.21	4.02	10.84	38.35	80.79
SimGRACE-($\geq$ 200)-(20)	0.07(**2.00x**)	0.05(**3.2x**)	0.03(**2.67x**)	0.12(**1.75x**)	0.86(**4.67x**)	1.82(**5.95x**)	4.17(**9.19x**)	7.79(**10.37x**)
MA-GCL-($\geq$ 200)-(20)	0.12(**1.17x**)	0.13(**1.23x**)	0.07(**1.14x**)	0.18(**1.17x**)	1.75(**2.29x**)	4.25(**2.55x**)	7.85(**4.89x**)	15.77(**5.12x**)
PerturbGCL-($\leq$ 100)-($\leq$ 10)	0.10(**1.40x**)	0.09(**1.77x**)	0.05(**1.60x**)	0.17(**1.23x**)	1.42(**2.83x**)	3.01(**3.60x**)	6.21(**6.17x**)	12.80(**6.31x**)

3.3.5 Efficiency Analysis

We compare the training cost of the PerturbGCL with strong GCL baselines (**RQ3**) in Table 3.4 and report the optimization curves in Figure 3.5 (right panel). We observed that ⑥ **PerturbGCL generally runs faster than augmentation-based and free baseline methods during training**. As seen in Table 3.4, although PerturbGCL runs slower than strong baseline (SimGRACE), PerturbGCL converges significantly faster than SimGRACE (see Figure 3.5 (Right)). As a result, the total training cost of PerturbGCL is lower. For example, SimGRACE takes $7.79 * 20 = 155.8$ seconds to train on RDT-M2K, while PerturbGCL takes $12.80 * 10 = 128$ seconds, which is close to 1.2 times speedup. These results demonstrate the efficiency of PerturbGCL compared with strong baseline methods.

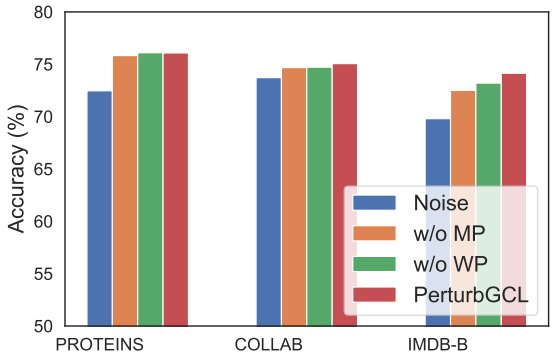

Figure 3.6: Ablation study of PerturbGCL on node benchmarks.

3.3.6 Ablation Study

We investigate the contributions of different components in PerturbGCL (**RQ4**). Figure 3.5 (left panel) and Figure 3.6 report the results on graph and node datasets, respectively. We observe that ⑦ **PerturbGCL benefits from the combination of randMP with weightPrune.** From the figures, PerturbGCL consistently outperforms two variants (i.e., w/o MP and w/o WP) in all cases, indicating the reciprocal effects of using *randMP* and *weightPrune* together. Moreover, ⑧ **replacing weightPrune with Gaussian noise, PerturbGCL drops significantly.** In both node and graph scenarios, PerturbGCL outperforms the "noise" variant with a great margin, verifying the effectiveness of the *weightPrune* proposal.

We also test the results of PerturbGCL under different contrastive objectives, such as Barlow Twins [98], Bootstrap [1], and InfoNCE [48]. From Figure 3.5 (middle), we observe that ⑨ **PerturbGCL performs generally better on InfoNCE and Barlow Twins objectives.** Given that InfoNCE is standard contrastive loss with negative samples and Barlow Twins is negative-sample free, PerturbGCL is ready to be applied on scenarios with informative negative sample or without negative samples by using different losses.

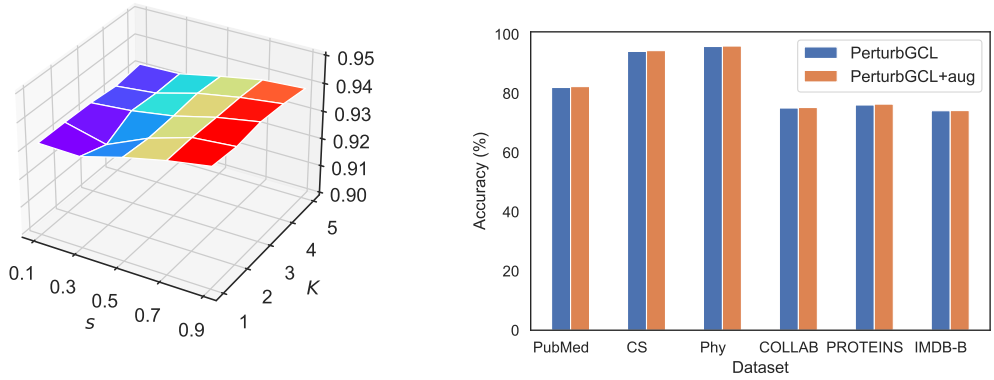

Figure 3.7: **Left:** Hyperparameter Analysis on Coauthor-CS. **Middle:** PerturbGCL with data augmentation.

3.3.7 Further Analysis

We finally investigate the sensitivity (**RQ5**) of PerturbGCL w.r.t. the propagation degree K and prune ratio s in Figure 3.7 (Left), and the impact of graph augmentation on PerturbGCL in Figure 3.7 (Right). We made two major observations.

(i) PerturbGCL performs stably when $K \in [1, 2, 3, 4, 5]$ and $s \in [0.7, 0.9]$. In Figure 3.7 (left), the performance of PerturbGCL when $K = 0.9$ (or 0.7) is consistently better than others. **(ii) PerturbGCL is complementary with advanced graph augmentation.** From Figure 3.7 (Middle), by feeding the augmented graphs as input, PerturbGCL can be further improved. However, the trade-off is that the improvement is not huge but the time to search optimal augmentation strategies is costing.

4. EFFICIENT FINE-TUNED GNN INFERENCE VIA RELIABLE KNOWLEDGE DISTILLATION

Given a fine-tuned GNN model with limited labeled samples, knowledge distillation [67] is the intuitive solution to boost its inference efficiency. However, directly leveraging soft labels from the GNN teacher is suboptimal since a large portion of unlabeled nodes will be incorrectly predicted by GNNs due to its limited generalization ability. To avoid the influence of mislabeled nodes, the common practice is to analyze their logit distributions from the teacher model [159, 160, 161]. For example, Zhang et al. [161] propose to assign larger weights to samples if their teacher predictions are close to one-hot labels. Zhu et al. [160] suggest filtering out data points if their teacher predictions mismatch with ground truth labels. Nevertheless, these methods cannot be applied in real-world graphs where node labels are expensive to access. Recently, Kwon et al. [159] suggest discriminating samples based on entropy values, by assuming that teacher predictions with lower entropy are more reliable. However, we found that entropy values are ineffective to distinguish the correct and wrong decision boundaries of GNN models since they are often largely overlapped, as we show in Figure 5.2 (right panel). Therefore, it still remains an open challenge to effectively distill semi-supervised GNN models to light-weight MLP students.

Motivated by this, we propose a novel KD framework – **RKD-MLP** to boost the MLP student via noise-aware distillation. It is noteworthy that while we focus on the MLP student for efficiency purposes, our solution is ready for other student types, such as GNNs. Specifically, RKD-MLP uses a meta-policy to filter out those unreliable soft labels by deciding whether each node should be used in distillation given its node representations. The student then only distills the soft labels of the nodes that are kept by the meta-policy. To train the meta-policy, we design a reward-driven objective based on a meta-set, where the meta-policy is rewarded for making correct filtering. The meta-policy is optimized with policy gradient to achieve the best expected reward and then be applied to unlabeled nodes. We iteratively update the meta-policy and the student model, achieving a win-win scenario: it substantially improves the performance of the vanilla MLP student by teach-

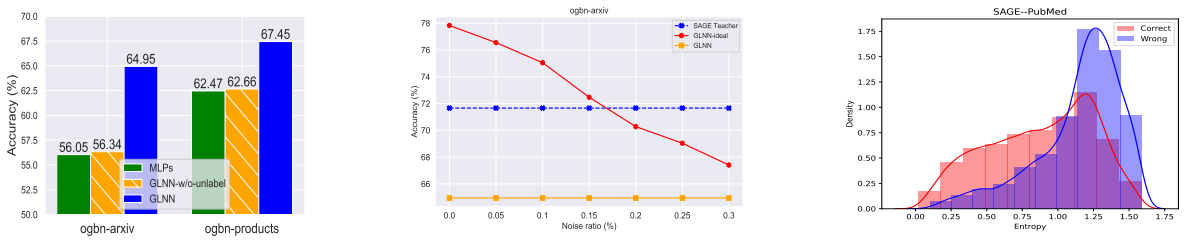

Figure 4.1: **Left:** The influence of unlabeled nodes on vanilla solution–GLNN [5]. GLNN-label is a variant of GLNN by excluding unlabeled nodes. **Middle:** The impacts of wrongly predicted nodes on the MLP student under different noise ratios. **Right:** Entropy distributions of wrongly and correctly predicted nodes by GNN teacher.

ing it with reliable guidance while maintaining the inference efficiency of MLPs without increasing the model size.

4.1 Problem Statement

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph with N nodes, where $\mathcal{V}$ and $\mathcal{E}$ stand for the node set and edge set, respectively. We use $\mathbf{X} \in \mathbb{R}^{N \times D}$ to denote node features, with row $\mathbf{x}_v$ being the D-dimensional feature vector of node $v \in \mathcal{V}$. We denote $\mathbf{Y} \in \mathbb{R}^{N \times C}$ as the label matrix with C classes of interest, where $\mathbf{y}_v \in \mathbb{R}^C$ represents the one-hot label vector of node v. In semi-supervised learning, which is a common task in graph analysis, we have a small portion of nodes being labeled while the majority of the nodes are unlabeled. We mark labeled nodes with superscript^L, i.e., $\mathcal{V}^L$, $\mathbf{X}^L$, and $\mathbf{Y}^L$, and unlabeled nodes with superscript^U, i.e., $\mathcal{V}^U$, $\mathbf{X}^U$, and $\mathbf{Y}^U$.

4.1.1 A Closer Look at Knowledge Distillation in Semi-Supervised Learning

To bridge the gap between vanilla MLPs and more advanced GNNs for graph analysis, an intuitive solution is to conduct cross-model knowledge distillation. Formally, let $\mathbf{z}_v \in \mathbb{R}^C$ denote the soft labels of node v predicted by a GNN teacher model, and $\hat{y}_v \in \mathbb{R}^C$ be the predictions of the MLP student model. The standard distillation process in [5] is expressed as:

$$\mathcal{L} = \lambda \sum_{v \in \mathcal{V}^L} \mathcal{L}_{CE}(\hat{\mathbf{y}}_v, \mathbf{y}_v) + (1 - \lambda) \sum_{v \in \mathcal{V}^L \cup \mathcal{V}^U} \mathcal{L}_{KL}(\hat{\mathbf{y}}_v, \mathbf{z}_v), \tag{4.1}$$

where $\mathcal{L}_{CE}$ is the standard cross-entropy loss on labeled nodes, while $\mathcal{L}_{KL}$ is **knowledge distillation loss**, i.e., the Kullback–Leibler divergence between the predictions of the MLPs student and GNNs teacher. λ is a trade-off parameter. Note that, different from supervised learning, the distillation loss in Eq. (4.1) naturally includes two parts in the semi-supervised scenario: labeled node set $\mathcal{V}^L$ and unlabeled node set $\mathcal{V}^U$. This design choice is inspired by standard semi-supervised learning philosophy, where unlabeled data is believed to be helpful in promoting model performance. According to our empirical results in Figure 5.2 (left), we observed that this tendency holds in KD. Without the soft labels of unlabeled nodes, the MLPs student can only perform comparably to the vanilla MLP baseline.

However, as aforementioned in the Introduction, we argue that blindly leveraging soft labels of all the nodes in Eq. (4.1) is suboptimal, since the soft labels from the teacher are noisy, especially for unlabeled nodes $\mathcal{V}^U$. Here, "noisy soft labels" refer to the soft labels of the nodes whose true labels mismatch the predictions of the GNN teacher. To verify this point, we conduct preliminary experiments from the oracle perspective, by assuming that the ground truths of unlabeled nodes are known. Then, we manually control the ratio of noisy soft labels in the knowledge distillation loss. Specifically, we use the same setting of GLNN to set up the MLP student, and then use the soft labels of correctly predicted nodes and a portion of wrongly predicted unlabeled nodes via random sample to train the MLP student. Figure 5.2 (middle) reports the results w.r.t. different ratios of wrongly predicted nodes.

Observation: **The soft labels of incorrectly predicted nodes restrict the capacity of MLPs student; By reducing the noise ratios, a stronger MLPs student can be easily achieved.** As shown in the middle panel of Figure 5.2, the MLP student's performance drops significantly as the noise ratio increases. If we can control the error ratio to some extent, e.g., 15%, the MLPs student can easily achieve comparable or even better results than GNNs teacher.

Nevertheless, it is a non-trivial task to effectively identify those wrongly predicted nodes from the correctly classified ones, following the standard entropy-based heuristic approach [159]. This is because the entropy distributions of the two groups are often largely overlapped in GNNs. For

example, the entropy distributions of wrongly and correctly predicted nodes are generally overlapped with 40% areas on different GNNs models as shown in Figure 5.2 (right).

The above observations pose a nature research question: *Can we filter out the noisy teacher guidance in an automatic fashion, such that a stronger MLP student can be achieved using reliable GNN knowledge?*

4.2 Methodology

In this part, we present a simple, generic, and effective KD framework to tackle the unreliable GNN guidance issues revealed in the Motivation section. Specifically, we first introduce the problem formulation in Section 4.2.1, and then elaborate on our proposal in Section 4.2.2.

4.2.1 Definition of Reliable Knowledge Distillation

Given a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, its feature matrix $\mathbf{X} \in \mathbb{R}^{N \times d}$, and label matrix $\mathbf{Y} \in \mathbb{R}^{N \times C}$. We use $\mathbf{Z} \in \mathbb{R}^{N \times C}$ to denote the soft label matrix produced by the teacher GNN model and f_{student} to denote a student model parameterized by multi-layer perceptrons. In traditional knowledge distillation settings [5], the student model is optimized according to two soft-label sets: (1) the labeled set $\mathcal{R}^L = \{(\mathbf{x}_v, \mathbf{z}_v) | v \in \mathcal{V}^L\}$, and (2) the unlabeled set $\mathcal{R}^U = \{(\mathbf{x}_v, \mathbf{z}_v) | v \in \mathcal{V}^U\}$. Nevertheless, as discussed in the Motivation section, using the soft labels of all nodes in $\mathcal{V}$ would degrade the performance of student models, since many unlabeled nodes are incorrectly predicted by the teacher model, which introduces unreliable guidance.

To this end, we study the *reliable knowledge distillation* (RKD) problem. The core idea of RKD is to filter out the wrongly predicted nodes by GNN teacher and construct a reliable soft-label set (i.e., $\mathcal{R}_r$) for student training. Formally, $\mathcal{R}_r = \mathcal{R}_r^L \cup \mathcal{R}_r^U$ consists of two parts, where $\mathcal{R}_r^L$ (or $\mathcal{R}_r^U$) includes those labeled (or unlabeled) nodes that are correctly predicted by the GNNs teacher. In practice, we can directly obtain the soft-label set $\mathcal{R}_r^L$ from labeled nodes since we already have the ground truths. Specifically, given a labeled node v, if the prediction from the teacher matches the ground truth, then $v \in \mathcal{R}_r^L$; otherwise, $v \notin \mathcal{R}_r^L$. Therefore, the main obstacle in RKD is how to determine the soft-label set $\mathcal{R}_r^U$ from unlabeled nodes, since no ground truth is available to check

their validity.

4.2.2 The Proposal

We present RKD-MLP, a general reinforced framework for training student MLPs via reliable GNN distillation. The full framework is illustrated in Figure 4.2. The key idea is to 1) learn a meta-policy network to determine the reliable soft label set and 2) train the student MLP based on the reliable guidance. After that, 3) an unified framework is designed to train the meta-policy network and student model jointly.

4.2.2.1 Meta Policy

To obtain the reliable soft label set $\mathcal{R}_r$, an intuitive solution is to utilize the uncertainty of teacher predictions [159]. For example, we can compute the entropy of all the nodes using their soft labels from the teacher GNNs, and then filter out those whose entropy values are higher than a pre-defined threshold. However, as shown in Figure 5.2 (Right), entropy can not well differentiate between the correct and incorrect nodes since they are largely overlapped. To overcome this limitation, we propose to develop a learning-based approach to automatically fit the complex decision boundary between them. Specifically, following [162], we assume that a meta-set with ground truth labels is available; in this work, we use the validation set as the meta-set. Then we propose to train a meta-policy with reinforcement learning (RL) to identify the reliable soft labels, where the meta-policy is updated based on the reward from the meta-set.

Formally, let $\mathcal{M}_{\text{meta}} = \{(\mathbf{x}_v, \mathbf{z}_v, \mathbf{y}_v, \mathcal{I}_v)\}_{v=1}^{m}$ denote a meta-set with m samples, where $\mathbf{z}_v$ is the teacher prediction and $\mathbf{y}_v$ denotes the ground truth. $\mathcal{I}_v = 1$ if the teacher model makes correct prediction; $\mathcal{I}_v = 0$ otherwise. We define the state, action and reward as follows. Let $\mathcal{S}$ be the state space; in this work, we use node representations (we will illustrate how to obtain these later) as the states, i.e., $\mathbf{x} \in \mathcal{S}$. Let $\mathcal{A} = \{0, 1\}$ be the action space, where 0 indicates that the soft label is unreliable, and 1 suggests that the soft label is reliable. Given a node $\mathbf{x}_v$, an agent takes action a_v and receives a scalar reward r_v, where a positive reward $r_v = 1$ is given if the label is indeed reliable (i.e., correct teacher prediction) when $a_v = 1$ or indeed unreliable (i.e., incorrect teacher

prediction) when $a_v = 0$, and $r_v = 0$ otherwise. Let $\pi : \mathcal{S} \rightarrow \mathcal{A}$ be a meta-policy that maps states to actions. With neural function approximators, we use π_θ to denote a parameterized meta-policy with parameters θ and $\pi_\theta(a|\mathbf{x}_v)$ to denote the probability of sampling a at state $\mathbf{x}_v$. The objective is to train the meta-policy network π_θ such that it can maximize the expected reward:

$$\mathcal{J}_{\text{meta}} = \mathbb{E}[r_v], \tag{4.2}$$

where node v is any node from all the nodes in the graph. Following the policy gradient theorem [163], we can calculate the gradient of $\mathcal{J}$ w.r.t. θ as

$$\begin{aligned} \bigtriangledown_\theta \mathcal{J}_{\text{meta}} &= \bigtriangledown_\theta \mathbb{E}[r_v] \\ &= \mathbb{E}[r_v \bigtriangledown_\theta \log \pi_\theta(a_v|\mathbf{x}_v)], \end{aligned} \tag{4.3}$$

where a_v is the currently selected action for node v. We approximate the above gradient with the samples in the meta-set $\mathcal{M}_{\text{meta}}$:

$$\bigtriangledown_\theta \mathcal{J}_{\text{meta}} \approx \sum_{v \in \mathcal{M}_{\text{meta}}} r_v \bigtriangledown_\theta [\log \pi_\theta(a_v|\mathbf{x}_v)], \tag{4.4}$$

where r_v can be obtained based on the ground truths in the meta-set by regarding the reliable soft labels as the ones that the teacher model makes correct predictions. The update of Eq. (4.4) can be unstable due to the high variance of the gradients. Thus, we introduce a baseline [164] for variance reduction. The final gradient can be written as

$$\bigtriangledown_\theta \mathcal{J}_{\text{meta}} \approx \sum_{v \in \mathcal{M}_{\text{meta}}} (r_v - B) \bigtriangledown_\theta [\log \pi_\theta(a_v|\mathbf{x}_v)], \tag{4.5}$$

where the baseline $B = (\sum_{v \in \mathcal{M}_{\text{meta}}} r_v)/m$ is the mean reward across the nodes in the meta-set, and $(r_v - B)$ can be interpreted as the advantage, i.e., the extent to which the current action is better than the average case. Then we use the meta-policy π_θ to predict the reliable soft labels for the unlabeled nodes. Specifically, the soft label of an unlabeled node u is considered reliable if the predicted

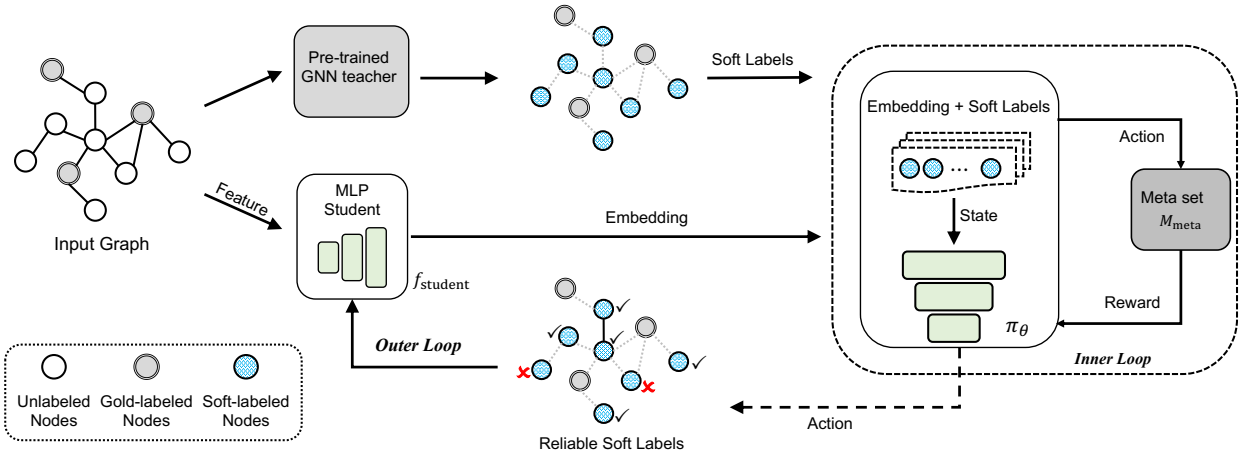

Figure 4.2: The RKD-MLP framework. Our meta-policy filters out noisy GNN teacher guidance, which is then used to train MLP student.

probability of $a = 1$ is larger than 0.5, i.e., $\mathcal{R}_r^U = \{(\mathbf{x}_u, \mathbf{z}_u) | \forall_u \in \mathcal{V}^U, \pi_\theta(a = 1|\mathbf{x}_u) > 0.5\}$.

Rationale. Despite the simplicity of RL in Eq. (4.5), the reward-driven objective enables the meta-policy to reason about the reliability of the soft labels based on node features. Once trained on the meta-set, the meta-policy can transfer to the unlabeled nodes to take the most rewarding action (i.e., reliable or unreliable) for each node.

4.2.2.2 Student Model Training with Reliable Guidance

By querying the meta-policy, we can train the student MLPs with better guidance. Formally, we rewrite Eq. (4.1) as:

$$\begin{aligned} \mathcal{L} = \lambda \sum_{v \in \mathcal{V}^L} \mathcal{L}_{CE}(f_{\text{student}}(\mathbf{x}_v), \mathbf{y}_v) + \\ (1 - \lambda) \sum_{v \in \mathcal{V}} \mathcal{I}_{\pi_\theta(a=1|\mathbf{x}_v) > \pi_\theta(a=0|\mathbf{x}_v)} \mathcal{L}_{KL}(f_{\text{student}}(\mathbf{x}_v), \mathbf{z}_v), \end{aligned} \tag{4.6}$$

$\mathcal{I}_{\pi_\theta(a=1|\mathbf{x}_v) > \pi_\theta(a=0|\mathbf{x}_v)}$ is an indicator function, which returns 1 if $\pi_\theta(a = 1|\mathbf{x}_v) > \pi_\theta(a = 0|\mathbf{x}_v)$; otherwise 0. That is, we only consider reliable soft labels identified by the meta-policy to compute the second term. Compared with traditional KD objective in Eq. (4.1), which leverages all soft labels from unlabeled nodes, the above training objective is noise-less, so the MLPs student model

Algorithm 4 Alternating Gradient Descent for RKD-MLP

Input: The meta-policy network π_θ, MLPs student f_{student}, the pre-trained GNN teacher

1: Initialize π_θ and f_{student}.
2: **while** *not converge* **do**
3: Obtain the node embedding $\mathbf{h}$ of unlabeled nodes based on MLP student.
4: Train the meta-policy network π_θ based on policy gradient in Eq. (4.5) and meta-set $\mathcal{M}_{\text{meta}}$.
5: Fix meta-policy π_θ and update the student model f_{student} based on reliable knowledge distillation loss in Eq. (4.6).
6: **end while**
7: **return** The well-trained MLP student f_{student}

will be trained with more reliable information from the teacher model.

4.2.2.3 The Unified Training Objective

Instead of training the meta-policy and student MLPs in a two-stage fashion, e.g., training the meta-policy first and then optimizing the student model, we propose to simultaneously train f_{meta} and f_{student} according to the following bi-level optimization framework:

$$\min_{f_{\text{student}}} \mathcal{L}(\mathbf{X}, \mathcal{Y}, f^*_{\text{meta}}) \quad \text{s.t.} \quad \pi^*_\theta := \arg\max_{\pi_\theta} \mathcal{J}_{\text{meta}}(\mathcal{M}_{\text{meta}}, f^*_{\text{student}}). \tag{4.7}$$

The outer objective $\mathcal{L}$ is defined in Eq. (4.6), which requires the meta-policy π_θ to select reliable soft-label predictions from the GNN teacher. The inner objective $\mathcal{J}_{meta}$ is defined in Eq. (4.5), and it takes node representations from the MLP student and soft label vectors from the GNN teacher as input. It is worth noting that the design of π_θ can take other node embeddings as input such as the raw features or hidden embedding from the GNN teacher. However, we find that using $\mathbf{h}_u$ as state representation is beneficial since jointly training the policy network and MLP student could reinforce their reciprocal effects. This is because learning a better MLP student requires π_θ to generate a more reliable soft label set while training a high-qualified policy needs more informative node embeddings as input. From Table 5.5 and 4.2, we can see that the MLP student performs better than the corresponding GNN teacher when it converges. Thus, it is reasonable to conjecture

that hidden representations of the MLP student are more informative. Following the common practice [53], we adopt the Alternating Gradient Descent (AGD) algorithm to optimize Eq. (4.7), by iteratively updating the outer and inner optimization objectives, as outlined in Algorithm 4.

4.3 Experiments

In this section, extensive experiments are reported to explore the following research questions.

- **RQ1:** How effective is RKD-MLP compared with state-of-the-art baselines in transductive and inductive settings?
- **RQ2:** Can RKD-MLP scale up to large-scale graphs?
- **RQ3:** What are the impacts of noisy node features or topology structures on RKD-MLP?
- **RQ4:** How effective is our meta-policy in identifying reliable teacher guidance?
- **RQ5:** How does each component of RKD-MLP contribute to the performance?
- **RQ6:** How efficient is RKD-MLP compared with other acceleration methods?

4.3.1 Datasets and Experimental Settings

Benchmark datasets. For comprehensive comparison, we use seven popular semi-supervised classification datasets with various scales and types, including **Cora**, **CiteSeer**, and **PubMed** [149], **WikiCS**, Amazon-Computers (**Compute**), Amazon-Photo (**Photo**), Coauthor-CS (**CS**) [165]. For experiments on large-scale graphs, we use two Open Graph Benchmark datasets [121]: **ogbn-arxiv** and **ogbn-products**.

Teacher GNNs. For a thorough comparison, we consider five promising GNNs architectures as teacher models in our knowledge distillation framework: GraphSAGE [22] (**SAGE**), **GCN** [21], **APPNP** [70], **GAT** [23], and **SGC** [109]. For extremely large-scale datasets such as ogbn-product, we consider two scalable teacher GNNs: **ClusterGCN** [62] and **GraphSAINT** [63].

Student competitors. In addition to the GNN teachers, we also include two types of student baselines for comparison. First, we consider three heuristic-based approaches: **Cluster**, **Entropy**,

and sample re-weighting (**SW**), which construct reliable soft-label set via clustering, relative prediction rankings, and sample re-weighting, respectively. Second, we include two MLPs based related work: vanilla **MLPs** and **GLNN** [5].

Transductive vs. Inductive. Follow previous studies [5], we evaluate our model under two node classification settings: transductive and inductive. The main difference between them is whether to use the test data for training or not. For the inductive setting, the test nodes as well as their edge links will not be used.

Implementation details. We build our model based on Pytorch and PyG library. For GNN teachers, following common practice in [52, 121], we employ a three-layer GNN encoder with dimension $d = 256$ for OGB benchmarks (ogbn-arxiv, and ogbn-products), while a two-layer GNN encoder with dimension $d = 128$ for other datasets. For MLP students, following [5], we set the number of layers and the hidden dimension of each layer to be the same as the teacher GNN. We set $\lambda = 0$ if not specified, since we empirically found that the proposed model is robust to λ as shown our preliminary experiments. All the experiments are run 5 times on GeForce RTX 2080 Ti GPUs, and we report the mean and the standard deviation.

4.3.2 How Effective is RKD-MLP Against other Baselines on Small Datasets?

Table 5.5& 4.2 report the results of our RKD-MLP with heuristic and MLPs based baselines (RQ1). We make three major observations. ① **Compared with vanilla MLPs and intuitive KD method - GLNN, RKD-MLP performs significantly better than them in all cases**. Specifically, RKD-MLP improves GLNN by up to 5.82% in the transductive setting (See Table 5.5). ②**RKD-MLP outperforms three heuristic solutions (SW, Entropy, and Cluster), especially on OGB datasets.** The possible explanation is that our meta-policy is trained end-to-end with the MLP student, so that they can reinforce their reciprocal effects. ③ **Compared with 5 GNN teachers, our proposal consistently achieves better results across different benchmark datasets and two evaluation scenarios**. Another interesting result is that the two heuristic methods (Entropy and Cluster) generally perform on par with or even better than GLNN across two settings. These results shed light on our motivation to study reliable knowledge distillation for MLP student training.

Table 4.1: Node classification accuracy on commonly used graph datasets in transductive learning. The best and second-best results are highlighted in Bold font and underlined, respectively.

Teacher	Student	Cora	CiteSeer	PubMed	WikiCS	Compute	Photo	CS
MLPs	-	58.04 ± 0.75	59.22 ± 1.31	70.54 ± 0.77	63.73 ± 1.51	67.80 ± 1.06	78.77 ± 1.74	84.80 ± 0.59
SAGE	-	79.70 ± 0.52	68.59 ± 0.27	76.55 ± 0.29	65.59 ± 0.88	82.97 ± 2.16	90.90 ± 0.84	90.56 ± 0.38
	SW	48.88 ± 6.81	54.47 ± 9.39	76.48 ± 0.59	58.77 ± 2.28	56.30 ± 3.31	63.15 ± 4.85	37.18 ± 1.22
	Entropy	80.73 ± 0.76	69.48 ± 0.52	78.48 ± 0.29	69.02 ± 1.41	83.53 ± 0.65	92.42 ± 0.74	91.79 ± 0.39
	Cluster	74.60 ± 0.35	$\mathbf{70.23 \pm 0.55}$	$\mathbf{80.66 \pm 0.07}$	67.05 ± 0.67	81.48 ± 0.95	87.84 ± 0.35	91.54 ± 0.15
	GLNN	80.00 ± 0.52	68.95 ± 0.70	76.97 ± 0.28	67.82 ± 1.36	83.04 ± 1.70	92.02 ± 1.15	90.95 ± 0.51
	RKD-MLP	$\mathbf{81.52 \pm 0.66}$	$\mathbf{70.23 \pm 0.48}$	$\mathbf{80.97 \pm 0.20}$	$\mathbf{71.77 \pm 0.64}$	$\mathbf{84.23 \pm 0.53}$	$\mathbf{93.78 \pm 0.46}$	$\mathbf{92.58 \pm 0.30}$
GCN	-	82.09 ± 0.28	69.62 ± 0.28	78.38 ± 0.27	67.29 ± 0.64	82.93 ± 0.67	91.09 ± 0.49	90.31 ± 0.25
	SW	52.24 ± 3.40	61.35 ± 2.00	75.91 ± 1.53	55.89 ± 0.70	54.01 ± 2.04	56.56 ± 5.07	35.93 ± 1.28
	Entropy	81.45 ± 0.27	69.95 ± 0.49	79.26 ± 0.24	70.10 ± 0.53	82.46 ± 0.79	92.78 ± 0.45	91.23 ± 0.38
	Cluster	74.32 ± 0.45	70.42 ± 0.33	80.90 ± 0.14	67.87 ± 0.43	81.63 ± 0.74	87.49 ± 0.26	91.66 ± 0.13
	GLNN	81.64 ± 0.90	69.86 ± 0.80	79.05 ± 0.30	69.43 ± 0.83	83.05 ± 0.72	92.12 ± 0.67	91.92 ± 0.52
	RKD-MLP	$\mathbf{82.53 \pm 0.16}$	$\mathbf{71.52 \pm 0.58}$	$\mathbf{81.61 \pm 0.39}$	$\mathbf{72.13 \pm 0.65}$	$\mathbf{84.44 \pm 0.57}$	$\mathbf{93.27 \pm 0.31}$	$\mathbf{92.88 \pm 0.15}$
APPNP	-	81.59 ± 0.64	70.44 ± 0.21	79.68 ± 0.19	67.84 ± 1.08	81.67 ± 1.21	91.92 ± 0.95	90.69 ± 0.28
	SW	51.02 ± 4.11	53.02 ± 3.63	77.92 ± 0.46	55.20 ± 1.76	56.06 ± 5.88	60.25 ± 4.27	35.67 ± 0.51
	Entropy	80.62 ± 0.37	70.37 ± 0.61	79.26 ± 0.25	68.97 ± 1.22	82.69 ± 1.35	92.16 ± 0.40	92.27 ± 0.26
	Cluster	75.14 ± 0.54	71.41 ± 0.35	80.59 ± 0.25	68.40 ± 0.54	81.10 ± 0.83	87.61 ± 0.48	91.64 ± 0.16
	GLNN	81.83 ± 0.78	70.67 ± 0.57	80.27 ± 0.36	69.87 ± 1.03	81.76 ± 1.11	91.92 ± 1.08	90.88 ± 0.40
	RKD-MLP	$\mathbf{82.80 \pm 0.49}$	$\mathbf{71.91 \pm 0.38}$	$\mathbf{81.37 \pm 0.50}$	$\mathbf{71.44 \pm 0.56}$	$\mathbf{83.06 \pm 1.01}$	$\mathbf{93.27 \pm 0.56}$	$\mathbf{92.74 \pm 0.22}$
GAT	-	82.64 ± 0.63	69.60 ± 0.42	77.80 ± 0.25	68.66 ± 0.97	81.90 ± 1.51	91.42 ± 0.74	89.73 ± 0.72
	SW	45.85 ± 7.67	47.00 ± 13.05	73.70 ± 2.63	51.82 ± 3.56	47.87 ± 13.00	62.23 ± 7.28	33.69 ± 5.14
	Entropy	80.67 ± 0.69	70.05 ± 1.24	79.11 ± 0.36	70.14 ± 1.21	82.26 ± 1.73	92.88 ± 0.70	89.64 ± 2.67
	Cluster	75.85 ± 0.52	70.39 ± 0.52	80.35 ± 0.30	67.96 ± 0.43	80.53 ± 1.40	87.46 ± 0.59	91.76 ± 0.19
	GLNN	82.55 ± 0.84	69.92 ± 0.31	78.81 ± 0.45	70.69 ± 1.17	82.19 ± 1.38	91.67 ± 0.62	91.00 ± 0.91
	RKD-MLP	$\mathbf{84.12 \pm 0.36}$	$\mathbf{72.56 \pm 0.68}$	$\mathbf{82.19 \pm 0.27}$	$\mathbf{73.21 \pm 0.84}$	$\mathbf{83.85 \pm 0.53}$	$\mathbf{93.49 \pm 0.66}$	$\mathbf{92.77 \pm 0.20}$
SGC	-	80.59 ± 0.47	69.59 ± 0.31	78.30 ± 0.14	67.78 ± 0.59	82.70 ± 0.64	91.30 ± 0.51	90.13 ± 0.38
	SW	46.80 ± 4.24	45.11 ± 14.58	75.84 ± 1.05	60.44 ± 1.90	54.11 ± 6.35	59.29 ± 3.59	35.98 ± 1.00
	Entropy	80.40 ± 0.40	70.34 ± 0.64	79.54 ± 0.15	69.79 ± 0.52	82.49 ± 1.17	92.22 ± 0.81	91.85 ± 0.48
	Cluster	74.60 ± 0.99	70.08 ± 0.70	80.69 ± 0.19	67.62 ± 0.91	81.60 ± 0.75	87.32 ± 0.70	91.90 ± 0.15
	GLNN	80.93 ± 0.75	69.87 ± 0.32	78.99 ± 0.16	69.87 ± 0.56	83.08 ± 0.70	92.54 ± 0.37	91.39 ± 0.65
	RKD-MLP	$\mathbf{82.99 \pm 0.34}$	$\mathbf{71.28 \pm 0.92}$	$\mathbf{81.72 \pm 0.40}$	$\mathbf{72.56 \pm 0.83}$	$\mathbf{84.56 \pm 0.89}$	$\mathbf{93.53 \pm 0.46}$	$\mathbf{93.02 \pm 0.40}$

4.3.3 How does RKD-MLP Perform on Large-Scale Graphs?

Figure 4.3 and Figure 4.5 summarize the results on two challenging large-scale graphs like ogbn-arixv and ogbn-products, from which we derive two insights (RQ2). ④ **RKD-MLP is scalable and can achieve much higher results than vanilla KD method**. As shown in Figure 4.3, RKD-MLP improves GLNN 8.5% and 6.3% on ogbn-arxiv and ogbn-products, respectively. ⑤ **Unlike small datasets, it is hard to train the MLP student on large graphs due to soft-label noises**. For instance, GLNN can achieve comparable results with GNN teacher on small datasets

Table 4.2: Node classification accuracy on commonly used graph datasets in inductive learning. The best and second-best results are highlighted in Bold font and underlined, respectively.

Teacher	Student	Cora	CiteSeer	PubMed	WikiCS	Compute	Photo	CS
MLPs	-	59.09 ± 2.96	59.60 ± 1.57	67.73 ± 1.23	62.55 ± 1.85	67.84 ± 1.78	79.44 ± 1.72	85.57 ± 0.92
SAGE	-	80.00 ± 0.42	71.79 ± 0.22	77.50 ± 0.19	65.58 ± 1.47	75.53 ± 1.61	87.13 ± 0.34	91.31 ± 0.36
	SW	45.81 ± 2.68	52.05 ± 17.53	78.56 ± 0.87	56.32 ± 2.34	46.74 ± 13.93	64.22 ± 6.53	34.60 ± 2.27
	Entropy	81.02 ± 0.36	73.20 ± 0.58	79.74 ± 0.37	67.78 ± 0.93	80.95 ± 0.83	91.35 ± 0.49	91.37 ± 0.10
	Cluster	72.98 ± 0.73	74.86 ± 0.80	80.20 ± 0.41	66.36 ± 0.57	$\mathbf{81.25 \pm 1.10}$	87.57 ± 0.44	91.24 ± 0.11
	GLNN	80.19 ± 0.71	72.31 ± 0.58	78.76 ± 0.58	67.29 ± 1.31	76.30 ± 1.65	87.70 ± 0.40	91.38 ± 0.53
	RKD-MLP	$\mathbf{81.69 \pm 0.96}$	$\mathbf{75.00 \pm 0.29}$	$\mathbf{81.18 \pm 0.65}$	$\mathbf{69.62 \pm 0.43}$	$\mathbf{81.50 \pm 0.77}$	$\mathbf{91.58 \pm 1.45}$	$\mathbf{92.41 \pm 0.54}$
GCN	-	80.29 ± 0.19	72.64 ± 0.45	78.88 ± 0.21	66.13 ± 0.15	80.33 ± 0.57	86.34 ± 0.57	89.02 ± 0.48
	SW	47.12 ± 4.07	59.08 ± 6.00	76.88 ± 0.81	54.40 ± 2.23	48.58 ± 10.25	62.39 ± 6.65	35.81 ± 0.54
	Entropy	80.87 ± 1.26	73.53 ± 0.77	79.28 ± 0.84	68.73 ± 0.70	$\mathbf{82.30 \pm 1.08}$	89.00 ± 0.83	91.59 ± 0.27
	Cluster	72.30 ± 0.52	$\mathbf{75.12 \pm 0.63}$	80.22 ± 0.32	65.63 ± 0.55	81.62 ± 0.68	87.30 ± 0.74	91.34 ± 0.17
	GLNN	79.23 ± 1.51	73.42 ± 0.41	79.36 ± 0.32	67.74 ± 0.49	80.56 ± 0.73	87.40 ± 0.57	90.64 ± 0.23
	RKD-MLP	$\mathbf{82.56 \pm 0.74}$	$\mathbf{75.22 \pm 0.71}$	$\mathbf{81.00 \pm 0.50}$	$\mathbf{69.88 \pm 0.38}$	$\mathbf{82.47 \pm 0.49}$	$\mathbf{90.36 \pm 1.22}$	$\mathbf{92.38 \pm 0.31}$
APPNP	-	82.87 ± 1.06	72.50 ± 0.71	79.18 ± 0.12	67.33 ± 0.88	80.24 ± 0.62	77.30 ± 1.94	89.61 ± 0.22
	SW	47.89 ± 3.89	59.22 ± 4.55	78.32 ± 0.94	54.46 ± 2.72	56.82 ± 4.46	61.88 ± 2.51	35.81 ± 2.63
	Entropy	79.61 ± 1.63	74.23 ± 0.64	80.54 ± 0.43	68.47 ± 0.84	81.20 ± 0.23	82.88 ± 0.84	91.79 ± 0.17
	Cluster	73.27 ± 0.52	74.23 ± 1.09	80.36 ± 0.42	66.21 ± 0.41	79.75 ± 0.81	80.61 ± 1.54	91.25 ± 0.10
	GLNN	82.68 ± 0.68	72.75 ± 1.08	79.88 ± 0.23	68.78 ± 1.00	80.22 ± 0.54	78.08 ± 1.91	89.99 ± 0.29
	RKD-MLP	$\mathbf{83.38 \pm 1.28}$	$\mathbf{74.93 \pm 0.79}$	$\mathbf{81.44 \pm 0.51}$	$\mathbf{69.83 \pm 1.01}$	$\mathbf{82.60 \pm 1.24}$	$\mathbf{85.10 \pm 0.90}$	$\mathbf{92.60 \pm 0.19}$
GAT	-	81.07 ± 0.79	72.72 ± 0.60	78.00 ± 0.28	67.77 ± 1.19	80.20 ± 1.38	90.38 ± 1.02	89.24 ± 0.86
	SW	43.68 ± 3.29	50.13 ± 9.22	76.98 ± 2.02	54.50 ± 2.58	57.40 ± 3.04	59.34 ± 4.60	34.44 ± 1.76
	Entropy	81.89 ± 1.36	74.09 ± 0.60	78.56 ± 1.13	69.99 ± 1.51	81.13 ± 2.40	91.83 ± 1.24	90.64 ± 0.65
	Cluster	73.85 ± 0.93	74.53 ± 0.57	80.36 ± 0.15	66.54 ± 0.50	80.58 ± 0.16	88.33 ± 0.57	90.15 ± 2.61
	GLNN	81.89 ± 0.39	72.94 ± 0.73	79.68 ± 0.62	70.10 ± 1.41	80.69 ± 0.26	91.31 ± 1.08	90.97 ± 0.97
	RKD-MLP	$\mathbf{82.61 \pm 1.30}$	$\mathbf{74.86 \pm 0.58}$	$\mathbf{82.28 \pm 0.40}$	$\mathbf{71.54 \pm 0.33}$	$\mathbf{81.97 \pm 0.39}$	$\mathbf{92.41 \pm 0.90}$	$\mathbf{92.86 \pm 0.26}$
SGC	-	66.15 ± 0.58	66.47 ± 0.74	75.60 ± 0.59	60.62 ± 0.26	72.70 ± 1.56	81.10 ± 0.75	87.03 ± 0.38
	SW	45.86 ± 8.28	46.77 ± 3.72	76.44 ± 1.98	57.01 ± 1.09	48.21 ± 8.94	63.64 ± 1.47	34.14 ± 2.34
	Entropy	68.23 ± 0.82	69.83 ± 0.99	78.32 ± 0.64	65.05 ± 1.18	78.11 ± 0.65	87.52 ± 0.64	89.98 ± 0.75
	Cluster	70.07 ± 0.66	71.39 ± 0.63	80.42 ± 0.19	65.42 ± 0.75	80.15 ± 0.70	84.56 ± 0.35	90.35 ± 0.07
	GLNN	66.54 ± 0.71	66.65 ± 0.36	76.70 ± 0.41	63.14 ± 1.12	75.48 ± 1.27	84.83 ± 2.60	88.09 ± 0.66
	RKD-MLP	$\mathbf{71.08 \pm 1.00}$	$\mathbf{71.45 \pm 1.49}$	$\mathbf{81.70 \pm 0.26}$	$\mathbf{66.04 \pm 0.88}$	$\mathbf{81.23 \pm 0.93}$	$\mathbf{89.03 \pm 0.68}$	$\mathbf{92.75 \pm 0.69}$

(See Table 5.5), but performs significantly worse on large graphs (See Figure 4.3& 4.5). By avoiding unreliable guidance, our RKD-MLP can easily outperform the GNN teacher on small datasets, and bridge the gap between GLNN and the GNN teacher on large graphs.

4.3.4 How Robust is RKD-MLP w.r.t. Feature or Topology Noises?

Figure 4.6 & 4.7 report the results of RKD-MLP on two types of noise scenarios (RQ3). In general, we observe that ⑥ **our proposal performs consistently better than other baselines, and is more robust on topology noise compared with feature noise**. For example, the performance

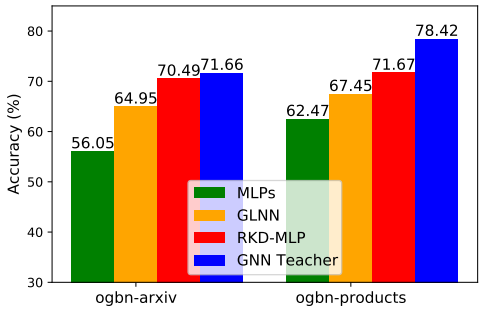

Figure 4.3: Accuracy results of RKD-MLP on large-scale graphs. **Left**: GraphSAGE teacher. **Right**: clustergnn teacher.

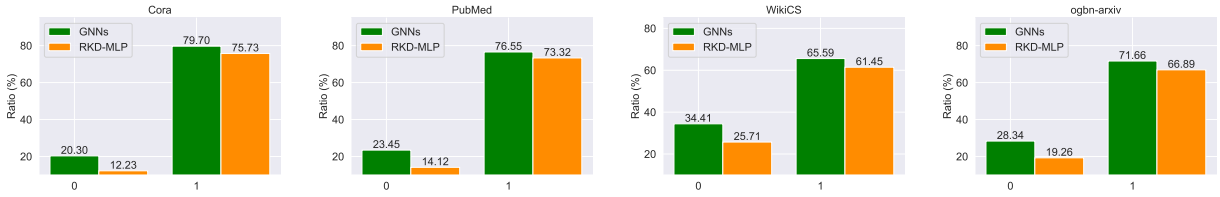

Figure 4.4: Visualization of the meta-policy's decisions. The x-axis represents two groups of unlabeled nodes, where 0 and 1 mean the GNN teacher makes the right and wrong predictions, respectively. The performance of RKD-MLP indicates how many nodes being wrongly (or correctly) classified by the GNN teacher are filtered out (or preserved) by the meta-policy.

gap between RKD-MLP and the second best baseline on incomplete graph structure (left two panels) is higher than that on noise feature (See Figure 4.6). We contribute this robustness gain to the proposed meta-policy, since it can filter out noisy teacher guidance.

4.3.5 How Effective is RKD-MLP in Eliminating Noisy Guidance?

We check the policy's action and visualize the results in Figure 4.4 (RQ4). As shown in the figure, ⑦ **our proposal can effectively detect nodes correctly predicted by the GNN teacher** while filtering out nodes being wrongly classified to some extent (the gap between in group "0"). For instance, RKD-MLP detects 93.34% unlabeled nodes being correctly predicted by the teacher while filtering out 67.96% nodes being wrongly predicted on the ogbn-arxiv dataset.

Table 4.3: Inference time (ms) on 10 randomly chosen nodes of two OGB datasets under the inductive setting. Numbers are copied from GLNN [5] since we have the same configuration.

Dataset	SAGE	QSAGE	PSAGE	Neighbor Sample	Ours
Arxiv	489.49	433.90 (1.13×)	465.43 (1.05×)	91.03 (5.37×)	**3.34 (146.55×)**
Products	2071.30	1946.49 (1.06×)	2001.46 (1.04×)	107.71 (19.23×)	**7.56 (273.98×)**

Table 4.4: Ablation study of RKD-MLP. clustergcn teacher for products while SAGE for others.

	Cora	CiteSeer	PubMed	WikiCS	Computer	Photo	CS	ogbn-arxiv	ogbn-products
RKD-MLP-iso	79.66	69.10	77.65	68.65	82.87	91.57	90.76	65.86	67.86
RKD-MLP-rand	79.22	68.25	76.32	67.35	82.28	91.16	90.03	64.52	66.88
RKD-MLP	**81.59**	**70.46**	**81.07**	**71.77**	**84.23**	**93.78**	**92.58**	**70.49**	**71.67**

4.3.6 Ablation Study

We study the importance of joint optimization and random selection on RKD-MLP with two ablations: RKD-MLP-iso and RKD-MLP-rand (RQ5). RKD-MLP-iso is obtained by separating the training of meta-policy and the MLP student. RKD-MLP-rand is obtained by replacing the meta-policy with random selection. We made two observations from Table 4.4. ⑧ **Jointly optimizing meta-policy and MLP student can reinforce their reciprocal effects, since RKD-MLP outperforms RKD-MLP-iso in all cases by a great margin**. Additionally, random selection fails to distinguish the decision boundary between correctly predicted samples and incorrectly classified samples by the GNN teacher, so it performs the worst.

4.3.7 Efficiency Analysis

To investigate the efficiency of our proposed model, we conduct experiments on two large-scale OGB datasets Arxiv and Products (RQ6). Following [5], we include three types of baselines, i.e., vanilla GNNs, common acceleration techniques (i.e., pruning and quantization), and sampling approaches. From Table 4.3 we observe that our distilled MLP student runs significantly faster than all other baselines. This is because our method reduces to a simple MLPs after distillation, while all baseline require to fetch neighboring nodes for inference. Given the high performance of

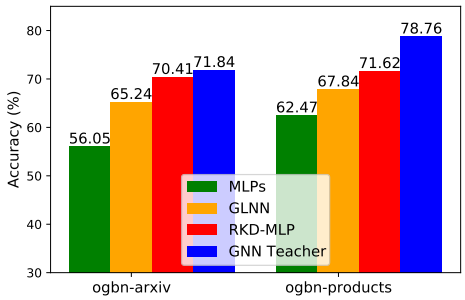

Figure 4.5: Accuracy results of RKD-MLP on large-scale graphs. **Left**: GCN teacher. **Right**: GraphSAINT teacher.

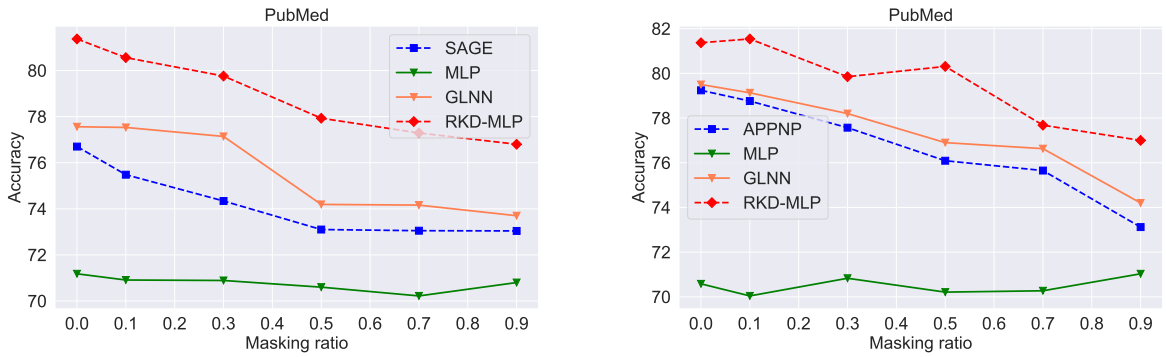

Figure 4.6: Accuracy results of RKD-MLP and other baselines w.r.t. noise graph topology.

RKD-MLP on Table 5.5 and 4.2, our model is more desired to be deployed on resource-sensitive applications.

4.3.8 The Influence of Parameter λ

In this section, we analyze the impacts of λ on our RKD-MLP model. Specifically, we vary λ from 0.1 to 1.0 with step size 0.1 and report the results on Figure 4.8.

From the figures, we can observe that our RKD-MLP model performs generally stable when $0.1 \leq \lambda \leq 0.9$. When $\lambda = 1.0$, it reduces to the vanilla MLP baseline, so the performance drops significantly.

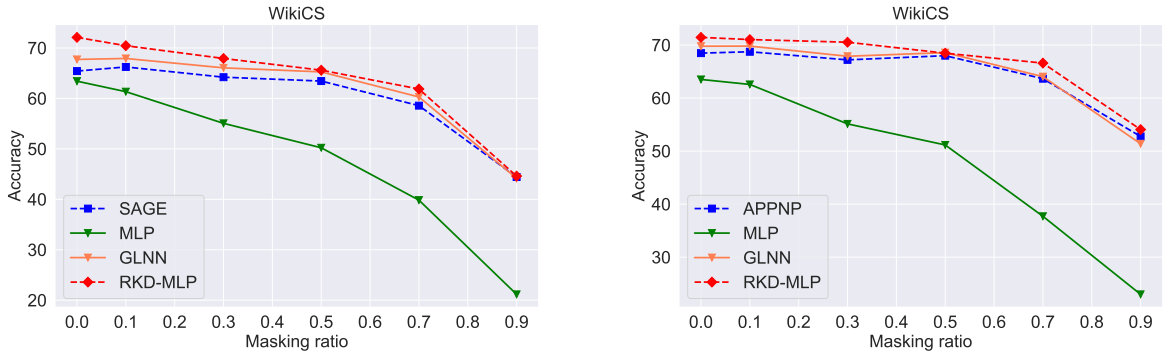

Figure 4.7: Accuracy results of RKD-MLP and other baselines w.r.t. noise node features.

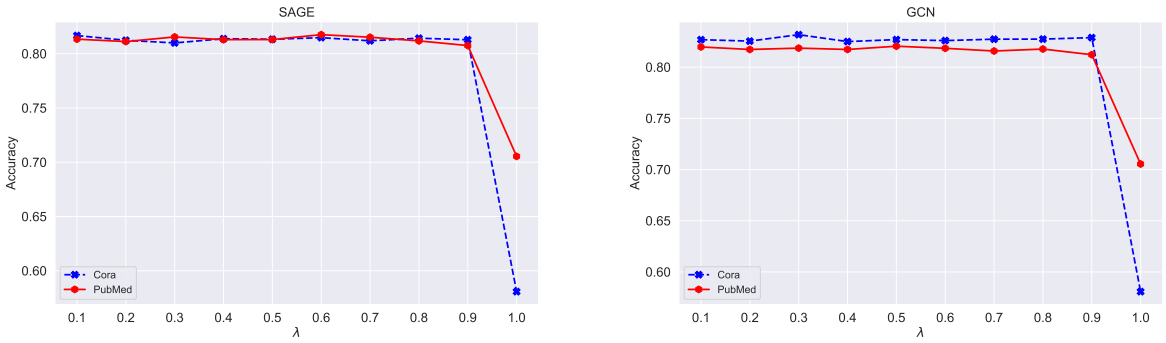

Figure 4.8: The impacts of trade-off parameter λ on RKD-MLP.

5. EFFICIENT GNN DEPLOYMENT WITH GENERALIZABLE GRAPH AUTOENCODERS *

Recently, some efforts [6, 107, 108, 166] have been devoted to advancing GAE for classification tasks. For example, GPT-GNN [107] builds an autoregressive framework to conduct node and edge reconstruction alternatively. GraphMAE [6] and GMAE [108] propose to reconstruct node features using a re-mask decoding strategy inspired by MAE [167]. These models can achieve competitive results with state-of-the-art graph contrastive learning (GCL) [60] methods on node-level and graph-level classification tasks in most cases. However, compared to traditional GAE models, GraphMAE and GMAE lose the ability to perform well on link prediction tasks, since they only focus on reconstructing node features, discarding the standard network structure reconstruction, which is crucial for inferring missing links [94, 103, 168].

To date, no GAE studies have succeeded in performing comparable results with GCL methods on node-level and graph-level classification scenarios without sacrificing their promise on link prediction tasks. This phenomenon casts doubt on the generalizability of GAEs as a universal graph learner, and raises several research questions: *Why traditional GAEs can not generalize well to graph classification tasks? How to build a generalizable GAE framework that could perform well on link-level, node-level, and graph-level learning tasks simultaneously?*

In this paper, we answer the above questions and identify three roadblocks to constructing generalizable pre-trained models with GAEs. ❶ The graph topology structure is over-emphasized. Most existing GAEs focus on accurately reconstructing the input graph structure (i.e., all pair-wise connections) to preserve the topological closeness between neighbors [94, 104, 105]. This strict requirement could be harmful to capturing the structure information among nodes [51], especially for real-world graphs which contain redundant [61] and noisy information [169]. ❷ Graph recon-

*Part of this chapter is reprinted with permission from "S2GAE: Self-Supervised Graph Autoencoders are Generalizable Learners with Graph Masking" by Qiaoyu Tan, Ninghao Liu, Xiao Huang, Soo-Hyun Choi, Li Li, Rui Chen, and Xia Hu, Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining, Pages 787–795, Copyright 2023 by Association for Computing Machinery.

struction without corruption is prone to overfit data. Prior GAEs usually adopt advanced neural encoders such as GNNs to learn node representations. Since the target links to be recovered are also explicitly utilized in GNNs for node embedding, it risks learning trivial solutions because GNNs are expressive under the homophily hypothesis. However, the masked autoencoders [167] that corrupt input and then attempt to recover it has shown great success in vision tasks, which might also be applicable to graphs. ❸ The decoder architectures are not expressive for edge reconstruction, especially after edge perturbation. Existing GAEs often parametrize their decoders with multi-layer perceptrons (MLPs) to estimate the similarity between nodes based on their latent representations. In traditional scenarios, an MLP-based decoder may be sufficient to model the correlations between nodes. However, if the graph structure is corrupted, especially when the non-trivial perturbation is conducted [167], the GNN encoder would inevitably be affected, leading to noisy node representations. In this case, the vanilla MLP decoder alone may not be robust enough to reconstruct edges. Notably, though some GAEs may have tackled one or two of the above issues individually, none of them deals with the three challenges as a whole.

Inspired by the aformentioned challenges, we present a principled framework, called Self-Supervised Graph Autoencoders (S2GAE), for generalizable graph representation learning. In a nutshell, S2GAE works by randomly masking a portion of the graph structure and then learning to reconstruct these missing edges with the unmasked graph structure. Compared with conventional GAEs, S2GAE also focuses on reconstructing the input graph but differs in the *model input*, *training target*, and *decoder architecture*. The key innovations lie in the new designs added to the three perspectives, which work together to unleash the generalization ability of GAEs, and equip S2GAE with broader applicability in practice. Table 5.1 compares the different design choices between GAEs and S2GAE. Specifically, S2GAE largely benefits from the following critical designs for their contributions to performance improvements.

5.1 Problem Statement

Standard GAE framework vs. S2GAE. Graph autoencoder [94, 103] is the typical framework to learn node representation in an unsupervised fashion. Assume $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is an undirected

Table 5.1: Comparison between generative SSL methods (GAE variants). *AE*: autoencoder framework; *Rec. Graph*: structure reconstruction objective; *Edge Perturb*: using edge masking to perturb input graph; *Direc. Mask*: direction-aware graph masking; *CC Dec.*: capturing cross-correlation between end nodes for edge reconstruction.

Methods	AE	Rec. Graph	Edge Perturb	Direc. Mask	CC Dec.
VGAE [94]	✓	✓	–	–	–
ARVGA [103]	✓	✓	–	–	–
LRR [166]	✓	✓	✓	–	–
GPT-GNN [107]	–	✓	✓	–	–
GMAE [108]	✓	–	–	–	–
GraphMAE [6]	✓	–	–	–	–
MaskGAE [170]	✓	✓	✓	–	–
S2GAE	✓	✓	✓	✓	✓

graph with n nodes, where $\mathcal{V}$ and $\mathcal{E}$ denote the sets of nodes and edges, respectively. Standard GAE aims to learn an encoder network $f : \mathcal{V} \times \mathcal{E} \rightarrow \mathbf{H} \in \mathbb{R}^{|\mathcal{V}| \times d}$ that maps each node $v \in \mathcal{V}$ into a d-dimensional embedding vector $\mathbf{h}_v \in \mathbb{R}^d$, and a decoder network $g : \mathbf{H} \rightarrow \mathcal{E}$ that reconstructs the network structure (e.g., edges in $\mathcal{E}$) from embeddings. The training objective of GAE is to reconstruct the input network structure as follows:

$$\mathbf{H} = f(\mathcal{V}, \mathcal{E}), \quad \mathcal{E}^r = g(\mathbf{H}). \tag{5.1}$$

$\mathcal{E}^r$ is the reconstructed network structure. It is worth noting that the reconstructed target could also be node attributes. However, in this work, we follow the generic setting [94] to reconstruct network structure, since node attributes are not always available in reality.

S2GAE. Although many GAE variants have been developed recently, they usually suffer from generalization challenges in applications beyond link prediction, such as node or graph classification. To bridge the gap, we present a novel GAE variant–S2GAE, which advances conventional GAE from self-supervised learning. Figure 5.1 depicts the overall architecture of S2GAE. Its key innovations lie in the direction-aware graph masking strategies and the cross-correlation decoder

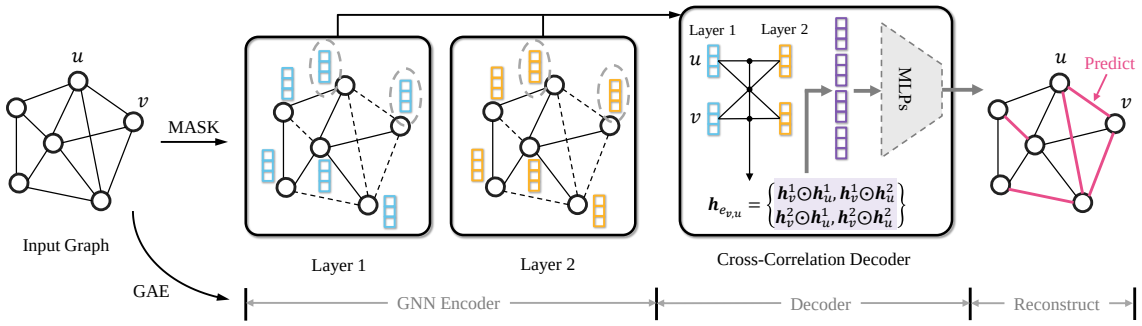

Figure 5.1: The proposed S2GAE architecture. Given a graph, we first apply direction-aware graph masking strategies to disturb it, obtaining perturbed graph and masked edge set. Then, the perturbed graph is fed into GNN encoder to produce hidden representations. Next, a tailored cross-correlation decoder is designed to reconstruct these masked edges by capturing the cross-correlation of their end nodes from multi-granularity representations. Finally, the whole framework is trained end-to-end by maximizing the likelihood of the masked edge set.

network. The former assists in generating effective and meaningful self-supervisory signals for input perturbation and target reconstruction, while the latter aids more accurate edge prediction by capturing the cross-correlation between two end nodes from their multi-granularity features. We summarize the differences between S2GAE and other GAE variants in Table 5.1.

In the following subsections, we will discuss our S2GAE method from *model input*, *encoder*, *decoder*, and *training objective* aspects.

5.2 Methodology

In this section, we present the proposed Self-Supervised Graph Autoencoder framework–S2GAE, which generalizes standard GAEs from the self-supervised learning perspective.

5.2.1 Perturbed Graph Input

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, standard GAE takes the complete graph $\mathcal{G}$ as input and learns to reconstruct the input network structure (i.e., $\mathcal{E}$) based on the similarity of encoded representations. However, as discussed above, such a design may over-emphasize the proximity information and lose sight of the structure information, leading to limited results on classification tasks. To address this issue, we propose to perturb the input graph structure and utilize only part of it as the input for

encoding. A similar idea has been successfully explored in the text [44] and image [167] fields. For example, BERT [44] adopts random dropping to generate partially observed word sequences for language modeling, and MAE [167] applies patch-aware random masking to yield masked image channels for visual representation.

Motivated by these, we propose to perturb the input graph via random dropping (i.e., graph masking). Formally, we randomly sample a subset of edges $\mathcal{E}_{mask}$ from the observed edges (i.e., $\mathcal{E}$) with some masking ratio ω, and then obtain the perturbed input graph $\mathcal{G}_{perb}$ as follows

$$\mathcal{G}_{perb} = (\mathcal{V}, \mathcal{E}_{remain}), \quad \mathcal{E}_{remain} = \mathcal{E} - \mathcal{E}_{mask}, \tag{5.2}$$

where $\mathcal{E}_{remain}$ denotes the remained edge set after graph masking. In practice, we adopt a uniform random sampling (with probability ω) without replacement to generate the masked edge set $\mathcal{E}_{mask}$. This is mainly because a uniform sampling distribution could prevent a potential center bias, i.e., the masked edge set is predominantly composed of influential nodes.

Behind Eq. (5.2), another thing that has remained so far is how to generate the masked edge set $\mathcal{E}_{mask}$. Similar to MAE [167], the intuitive solution is to treat nodes as pixels and then uniformly sample neighboring nodes for edge masking. That is, we regard edges as undirectional as done by graph contrastive methods [48]. This implementation is reasonable for images because they are dense and grid-like data. However, it might be suboptimal to real-world graphs, especially for large-scale data, since they are often sparse [171]. Therefore, it is desirable to consider graph characteristics when applying edge masking. To this end, we suggest two elegant direction-aware graph masking strategies that align with our proposal.

I. **Undirected masking (UM).** We treat the link between node v and v as undirected. That is, $e_{v,u}$ and $e_{u,v}$ are regarded as the same, and only one copy is included in $\mathcal{E}$. Therefore, after performing random sampling over $\mathcal{E}$, the resultant masked edge set $\mathcal{E}_{mask}$ and $\mathcal{G}_{perb}$ are also undirected.

II. **Directed masking (DM).** We treat links in the graph as directed (even for undirected graphs).

Hence, $e_{v,u}$ and $e_{u,v}$ are regarded as different, and both of them are stored in $\mathcal{E}$. Deleting $e_{v,u}$ does not mean $e_{u,v}$ is also deleted. After performing random sampling over $\mathcal{E}$, the resultant masked edge set $\mathcal{E}_{mask}$ and $\mathcal{G}_{perb}$ are directed.

Despite being conceptually simple, the two proposed graph masking strategies perform surprisingly well in our experiments. Specifically, we found that the best graph masking strategy relates to graph statistics and downstream tasks. For example, if the network is dense or known to have redundant information, e.g., social networks, it is better to adopt the tougher strategy– undirected masking. In contrast, the directed masking might be a better choice if the original graph is sparse. We empirically verify these observations and provide more insights in Section 5.3.5. Notably, existing SSL studies [3, 48, 170] on graphs all belong to the UM strategy; we are the first to consider DM schema for graph masking.

5.2.2 GNN Encoder

To effectively map nodes into hidden representations, GNNs [21, 22] are often applied as the encoder backbone. The goal of GNNs is to update node representation by leveraging representations from itself and its neighboring nodes, expressed as:

$$\mathbf{h}_v^{(k)} = \text{COM}(\mathbf{h}_v^{(k-1)}, \text{AGG}(\{\mathbf{h}_u^{(k-1)} : u \in \mathcal{N}_v\})), \tag{5.3}$$

where $\mathbf{h}_v^{(k)} \in \mathbb{R}^d$ denotes the embedding of node v at the k-th layer, and $\mathcal{N}_v = \{u|u|e_{v,u} \in \mathcal{E}\}$ is a set of direct neighbors of node v, with $\mathbf{h}_v^{(0)} = \mathbf{x}_v$. The function AGG is used to aggregate features from neighbors [21], and function COM is used to combine the aggregated neighbor information and its own node embedding from the previous layer. For a GNN encoder with K layers, there are K node representations $\{\mathbf{h}_v^{(1)}, \mathbf{h}_v^{(2)}, \cdots, \mathbf{h}_v^{(K)}\}$ being generated, where $\mathbf{h}_v^{(k)}$ captures the neighborhood structure within k hops.

Difference. Unlike traditional GAEs that take the original graph $\mathcal{G}$ as input in Eq. (5.3), our S2GAE feeds the perturbed graph $\mathcal{G}_{perb}$ to the GNN encoder. Therefore, the encoded node representations $\{\mathbf{h}_v^{(1)}, \mathbf{h}_v^{(2)}, \cdots, \mathbf{h}_v^{(K)}\}$ would inevitably contain noises especially when ω is large. Thus

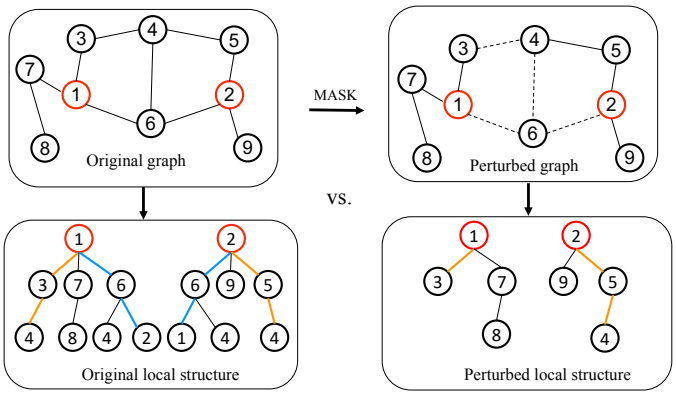

Figure 5.2: *Upper:* the original input graph vs. the perturbed input graph. Dashed lines indicate edges being masked out. *Lower:* the complete local structure for the GNN encoder vs. the incomplete local structure after perturbation. Paths highlighted with "orange" and "blue" colors indicate important motifs for the formation of link between nodes 1 and 2.

we develop a specific cross-correlation decoder introduced in the next section to combat this issue and improve the reconstruction capacity.

5.2.3 Cross-Correlation Decoder

Given an edge (v, u) and their hidden representations $(\mathbf{h}_v^{(K)}, \mathbf{h}_u^{(K)})$, existing GAEs [94, 22, 105] often define the decoder network as either the inner product of their embeddings (i.e., $g(v, u) =< \mathbf{h}_v^{(K)}, \mathbf{h}_u^{(K)} >$) or an MLP layer built upon the concatenation of their embeddings, i.e., $g(v, u) = \text{MLP}(\mathbf{h}_v^{(K)} || \mathbf{h}_u^{(K)})$. However, as discussed in Section 5.2.2, the representation quality is impaired when we use the perturbed graph $\mathcal{G}_{perb}$ as input, especially when non-trivial graph masking (e.g., with a relatively high masking ratio) is performed. Consequently, directly applying standard decoder architecture is rather limited. Meanwhile, this limitation is caused by the incomplete graph structure after masking, so merely adopting more advanced neural networks such as GNNs will not solve the problem, since GNNs depend on reliable network structure for message propagation. This is further validated through experiments in our ablation study (see Section 5.3.5).

An illustration of the above challenge is provided in Figure 5.2. The missing link between nodes 1 and 2 can be originally inferred via two motifs: the triangle between nodes 1, 6, and 2, and the pentagon among nodes 1, 3, 4, 5, and 2. In a normal case, the anchor edge is easily

predicted by information about either the orange or blue path in the last hidden layer of the GNN encoder. After perturbation, however, the last hidden layer may not learn useful information, since the information path is incomplete and includes no-negligible unrelated information flows (e.g., ①−⑦−⑧ and ②−⑨). In this case, a better choice is to focus on the ①–③ and ②–⑤–④ paths (shown in the bottom right figure) and activate the pentagon motif–①–③–④–⑤–②. From the GNN's perspective, it means we enforce the similarity between the first-layer and second-layer hidden representations of node ① and ②, respectively. That is, we need to model the cross-correlation between the 1-hop neighbors of node ① and the 2-hop neighbors of ②.

In light of this, we propose a novel cross-correlation decoder to explicitly capture this beneficial cross-correlation similarity between two end nodes in different granularities. Formally, given the K hidden representations of node v and u, i.e., $\{\mathbf{h}_v^{(k)}, \mathbf{h}_u^{(k)}\}_{k=1}^{K}$, we generate its cross representation $\mathbf{h}_{e_{v,u}}$ as bellow.

$$\mathbf{h}_{e_{v,u}} = ||_{k,j=1}^{K} \mathbf{h}_v^{(k)} \odot \mathbf{h}_u^{(j)}, \tag{5.4}$$

where $\odot$ denotes the element-wise product, and $\mathbf{h}_{e_{v,u}} \in \mathbb{R}^{dK^2}$ is the final edge representation. The multiplication between the k-th and j-th embedding vectors of nodes v and u–$\mathbf{h}_v^{(k)} \odot \mathbf{h}_u^{(j)}$, is an effective manner to model their cross-correlations: highlighting common properties and diluting discrepant information, thanks to the property of the element-wise product. As a result, most of the elements in $\mathbf{h}_{e_{v,u}}$ will be zero or small values, and only those elements that are highly correlated between the two nodes are reserved, which helps the follow-up edge prediction.

After obtaining the cross-correlation representation of edge (v, u), we adopt an MLP layer to predict its existent probability via $g(v, u) = \text{MLP}(\mathbf{h}_{e_{v,u}}) \in \mathbb{R}$, which gives rise to the name of cross-correlation decoder. We want to remark that another popular choice in the GNN community is to concatenate all intermediate representations for node embedding. That is, we first obtain v's (or u's) representation by combining all its intermediate embeddings, like $\hat{\mathbf{h}}_v = ||_{k=1}^{K} \mathbf{h}_v^{(1)}$ (or $\hat{\mathbf{h}}_u = ||_{k=1}^{K} \mathbf{h}_u^{(1)}$), and then combine them for edge prediction, i.e., $g(v, u) = \text{MLP}(\hat{\mathbf{h}}_v || \hat{\mathbf{h}}_u)$. This strategy, at first glance, seems effective in preserving all information between two end nodes. However, in our edge masking scenario, its effectiveness will be substantially offset due to the

incomplete graph structure $\mathcal{G}_{preb}$, as each hidden representation is noisy. As a result, it's difficult for the edge predictor–$g(\cdot)$ to make the decision from them. By contrast, our cross-correlation decoder can filter that inconsistent and unnecessary information out and help $g(\cdot)$ make predictions based on these common and informative features. We empirically verify their differences in the ablation study section 5.3.5.

5.2.4 Why is S2GAE Generalizable?

In traditional GAE methods, the learning task is reconstructing the observed edges in $\mathcal{E}$. However, this training objective may tend to over-fit the proximity information [51], damaging the generalization ability of GAEs. In this paper, we propose to predict the masked edges in $\mathcal{E}_{mask}$ in training, inspired by the success of masked autoencoding in computer vision [167]. We believe this learning paradigm is useful for GAE, since it enforces the model to understand the topological structure among nodes. For example, to estimate the missing link between nodes 1 and 2 in Figure 5.2 after masking, the model needs to recognize the pentagon structure among nodes 1, 3, 4, 5, and 2.

Formally, let $g(v, u) = g(f(v), f(u))$ be the estimated link probability of nodes v and u, the training objective of S2GAE is to reconstruct the masked edges:

$$\mathcal{L} = -\frac{1}{|\mathcal{E}_{mask}|} \sum\nolimits_{(v,u)\in\mathcal{E}_{mask}} \log \frac{\exp(g(v,u))}{\sum_{z\in\mathcal{V}} \exp(g(v,z))}. \tag{5.5}$$

In practice, the summation operation in the denominator of Eq. (5.5) is often approximated by negative sampling [171] to accelerate the training. We will show that, although the above loss function is the same as traditional GAE losses [94, 22], by applying the masked-autoencoder training paradigm, the proposed framework could train more generalizable models.

Connection to contrastive learning. We analyze the theoretical connection between our S2GAE and graph contrastive learning [1, 52] to explain why S2GAE could train generalizable representations. In a nutshell, we found that our S2GAE actually optimizes an edge-aware contrastive learning objective, thanks to the design of *perturbing graph input* and *reconstructing*

masked edges.

Specifically, let $\mathbf{h}_{v,i}$ and $\mathbf{h}_{v,j}$ denote representations of v under two kinds of augmentations strategies: i and j. The standard node-level contrastive learning loss [48] is formulated as:

$$\ell_v = -\log \frac{\exp(\text{sim}(\mathbf{h}_{v,i}, \mathbf{h}_{v,j})/\tau))}{\sum_{v'=1,v'\neq v}^{N} \exp(\text{sim}(\mathbf{h}_{v,i}, \mathbf{h}_{v',j})/\tau))}, \tag{5.6}$$

where $\text{sim}(\cdot)$ denotes the similarity function, such as the dot product, and N is the number of negative samples. τ is a temperature parameter. According to GraphCL [48], the key insight of Eq. (5.6) is to generate two representations of the same node after creating two views with different graph augmentation operations. In light of this, we can regard each edge (v, u) as a virtual node and represent its two augmented views based on the end nodes, i.e., the perturbed local subgraphs of nodes v and u. Then, we can rewrite Eq. (5.5) to its contrastive form as:

$$\ell_{v,u} = -\log \frac{\exp(g(f(\mathcal{G}^v_{perb}), f(\mathcal{G}^u_{perb})))}{\sum_{z=1,z\neq v}^{N} \exp(g(f(\mathcal{G}^v_{perb}), f(\mathcal{G}^z_{perb})))}, \tag{5.7}$$

Here, we misuse notation $\mathcal{G}^v_{perb}$ to denote the augmented subgraph of node v after edge masking. $f(v)$ is a GNN encoder that generates v's representation by using its perturbed local subgraph. We could find that Eq. (5.7) is a special implementation of Eq. (5.6) by: 1) designing the similarity function with $g(\cdot)$, 2) reducing two graph augmentation operations to one global edge masking perturbation, and 3) treating an anchor edge as a virtual node and treating local subgraphs of two end nodes as two augmented views. To summarize, the edge reconstruction training in graph autoencoder guarantees its effectiveness on link prediction task. Meanwhile, the edge-wise contrastive nature of our S2GAE framework sheds light on its generalizability to other tasks, such as classification.

5.2.5 Training and Inference

S2GAE is optimized via stochastic gradient descent as shown in Algorithm 5. It first perturbs the input graph $\mathcal{G}$ using our direction-aware graph masking proposals to obtain the perturbed graph

$\mathcal{G}_{perb}$ and masked edge set $\mathcal{E}_{mask}$. Then, the perturbed graph $\mathcal{G}_{perb}$ will be fed into the GNN encoder $f(\cdot)$ to generate node representations. Next, the cross-correlation decoder $g(\cdot)$ estimates the existence probability of anchor edges based on multi-scale representations of their end nodes. Finally, the model is trained end-to-end by maximizing the likelihood of masked edges in $\mathcal{E}_{mask}$.

After training, our S2GAE model can be applied to different downstream tasks, such as link prediction, node classification, and graph classification, by feeding the original graph without masking to generate node representations. Specifically, for classification, we directly use the learned node representations for node-level classification, while adopting a non-parameterized readout function (e.g., MaxPooling and MeanPooling) to obtain graph embedding for graph-level classification. For link prediction, given an unseen edge, we estimate its existence probability by further feeding the representations of its end nodes to the decoder $g(\cdot)$. Based on different application recipes, we empirically show that our S2GAE model can achieve competitive results across the three critical tasks.

Algorithm 5 Self-Supervised Graph Autoencoder (S2GAE)

Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, GNN encoder depth K, masking ratio ω, embedding dimension d;

while *not converge* **do**

  Perturb input graph $\mathcal{G}$ with direction-aware graph masking (UM or DM) to generate the perturbed graph $\mathcal{G}_{perb}$ and masked edge set $\mathcal{E}_{mask}$;
  Perform GNN encoding on the perturbed graph $\mathcal{G}_{perb}$ according to Eq. (5.3);
  Compute the cross representations of edges in $\mathcal{E}_{mask}$ according to Eq. (5.4);
  Update the GNN encoder $f(\cdot)$ and cross-correlation decoder $g(\cdot)$ by minimizing the reconstruction loss defined on $\mathcal{E}_{mask}$ according to Eq. (5.5);

Return The trained S2GAE model.

5.3 Experiments

In this section, we conduct extensive experiments to benchmark the generalization ability of S2GAE over three important graph learning tasks: link prediction, node classification, and graph classification.

5.3.1 Datasets and Experimental Settings

Datasets. We include 14 node-level benchmark datasets (See Table 5.2 for statistics) and 7 graph-level benchmark datasets (See Table 7 for statistics) for experiments. These datasets are publicly available online and have been frequently adopted by other researchers.

Table 5.2: Dataset statistics of node-level benchmarks.

Data	# Nodes	# Edges	# Features	Split ratio	# Classes
Cora	2,708	5,429	1,433	85/5/15	7
CiteSeer	3,312	4,660	3,703	85/5/15	6
PubMed	19,717	44,338	500	85/5/15	3
BlogCatalog	5,196	171,743	8,189	85/5/15	–
Flickr	7,575	239,738,	12,047,	85/5/15	–
ogbl-ddi	4,267	1,334,889	-	80/10/10	–
ogbl-collab	235,868	1,285,465	128	92/4/4	–
ogbl-ppa	576,289	30,326,273	58	70/20/10	–
Amazon-Computers	13,752	245,861	767	–	10
Amazon-Photo	7,650	119,081	745	–	8
Coauthor-CS	18,333	81,894	6,805	–	15
Coauthor-Physics	34,493	247,962	8,415	–	5
ogbn-arxiv	169,343	1,166,243	128	–	40
ogbn-proteins	132,534	39,561,252	8	–	112

Table 5.3: Dataset statistics of graph-level benchmarks.

Data	# graphs	Avg. # nodes	# classes
IMDB-B	1,000	19.8	2
IMDB-M	1,500	13.0	3
PROTEINS	1,113	39.1	2
COLLAB	5,000	74.5	3
MUTAG	188	17.9	2
REDDIT-B	2,000	429.7	2
NCI1	4,110	29.8	2

Table 5.4: Link prediction results on both contrastive and generative methods. The results are not reported due to unavailable code or out-of-memory.

	Cora AUC	CiteSeer AUC	PubMed AUC	Blog. AUC	Flickr AUC	ogbl-ddi Hits@20	ogbl-collab Hits@50	ogbl-ppa Hits@10	A.R.
DGI	90.02 ± 0.80	95.53 ± 0.40	91.24 ± 0.60	74.85 ± 1.46	92.44 ± 0.30	–	–	–	7.60
GIC	93.54 ± 0.60	97.04 ± 0.50	93.71 ± 0.30	68.77 ± 2.68	86.71 ± 0.46	–	–	–	8.00
MVGRL	87.46 ± 0.38	88.95 ± 0.66	88.36 ± 0.59	73.80 ± 1.72	91.40 ± 0.36	–	–	–	10.60
BGRL	87.08 ± 0.24	85.82 ± 0.36	96.75 ± 0.12	73.56 ± 1.46	88.93 ± 0.27	–	21.58 ± 1.92	–	8.50
GAE	91.09 ± 0.01	90.52 ± 0.04	96.40 ± 0.01	84.91 ± 1.44	92.50 ± 0.40	37.07 ± 5.07	44.75 ± 1.07	2.52 ± 0.47	5.25
GraphSage	86.33 ± 1.06	85.65 ± 2.56	89.22 ± 0.87	78.62 ± 0.72	89.70 ± 0.14	53.90 ± 4.74	54.63 ± 1.12	1.87 ± 0.67	7.62
ARGE	92.40 ± 0.00	91.94 ± 0.00	96.81 ± 0.00	75.48 ± 0.25	87.73 ± 0.19	20.43 ± 4.66	28.39 ± 2.51	0.41 ± 0.26	7.12
GPT-GNN	92.28 ± 0.31	91.36 ± 0.66	97.83 ± 0.03	86.16 ± 0.14	92.54 ± 0.09	37.05 ± 5.96	42.41 ± 1.80	1.57 ± 0.94	4.37
RRL	88.46 ± 1.85	85.47 ± 1.01	93.10 ± 0.49	83.64 ± 0.69	93.13 ± 0.08	16.84 ± 2.23	29.88 ± 2.94	0.24 ± 0.19	7.62
GraphMAE	89.19 ± 0.00	91.20 ± 0.11	93.72 ± 0.00	76.60 ± 1.32	88.69 ± 0.04	–	22.79 ± 1.62	0.18 ± 0.28	7.75
MaskGAE	$\mathbf{96.66 \pm 0.17}$	$\mathbf{98.00 \pm 0.23}$	$\mathbf{99.06 \pm 0.05}$	81.06 ± 3.06	93.60 ± 0.14	16.25 ± 1.60	32.47 ± 0.59	0.23 ± 0.04	4.00
S2GAE-SAGE	95.05 ± 0.76	94.85 ± 0.49	97.38 ± 0.17	80.84 ± 1.15	92.32 ± 0.24	$\mathbf{66.00 \pm 9.49}$	49.27 ± 0.96	1.37 ± 0.38	4.00
S2GAE-GCN	93.52 ± 0.23	93.29 ± 0.49	98.45 ± 0.03	$\mathbf{87.06 \pm 0.37}$	$\mathbf{94.38 \pm 0.02}$	65.91 ± 3.50	$\mathbf{54.74 \pm 1.06}$	$\mathbf{3.98 \pm 1.33}$	2.13

Table 5.5: Node classification performance on both contrastive and generative methods. A.R. is the average rank.

Method	Cora	CiteSeer	PubMed	Compute	Photo	CS	Phy	arxiv	proteins	A.R.
DGI	85.41 ± 0.34	74.51 ± 0.51	85.95 ± 0.66	84.68 ± 0.39	91.57 ± 0.25	92.77 ± 0.38	94.55 ± 0.13	67.08 ± 0.43	50.31 ± 0.55	6.44
GIC	$\mathbf{87.70 \pm 0.01}$	$\mathbf{76.39 \pm 0.02}$	85.99 ± 0.13	82.50 ± 0.22	90.65 ± 0.47	91.33 ± 0.30	93.49 ± 0.42	64.00 ± 0.22	48.55 ± 0.47	6.66
MVGRL	85.86 ± 0.15	73.18 ± 0.22	84.86 ± 0.31	88.70 ± 0.24	92.15 ± 0.20	92.87 ± 0.13	95.35 ± 0.08	68.33 ± 0.32	–	5.12
BGRL	86.16 ± 0.20	73.96 ± 0.14	86.42 ± 0.18	90.48 ± 0.10	93.22 ± 0.15	93.35 ± 0.06	$\mathbf{96.16 \pm 0.09}$	71.77 ± 0.19	–	3.00
GCN	83.60 ± 0.52	63.37 ± 1.21	78.23 ± 1.63	87.21 ± 0.26	90.05 ± 0.05	89.79 ± 0.09	93.26 ± 0.05	66.01 ± 0.37	61.67 ± 0.35	8.44
GraphSage	74.30 ± 1.84	60.20 ± 2.15	81.96 ± 0.74	87.05 ± 0.25	89.29 ± 0.17	89.74 ± 0.19	93.35 ± 0.06	64.79 ± 2.91	55.39 ± 0.79	9.22
ARGVA	85.86 ± 0.72	73.10 ± 0.86	81.85 ± 1.01	83.36 ± 0.43	86.55 ± 0.31	84.68 ± 0.26	92.89 ± 0.11	50.06 ± 1.21	40.73 ± 0.68	9.77
GPT-GNN	84.69 ± 0.09	71.82 ± 0.13	81.45 ± 0.18	83.06 ± 0.12	89.15 ± 0.28	91.07 ± 0.21	95.02 ± 0.15	70.16 ± 0.10	61.45 ± 0.23	7.77
RRL	57.29 ± 0.13	59.57 ± 1.77	75.06 ± 0.37	80.49 ± 0.10	82.66 ± 0.24	84.71 ± 0.95	94.90 ± 0.02	66.36 ± 0.13	60.26 ± 0.05	10.22
GraphMAE	85.45 ± 0.40	72.48 ± 0.77	85.74 ± 0.14	88.04 ± 0.61	92.73 ± 0.17	$\mathbf{93.47 \pm 0.04}$	96.13 ± 0.03	71.86 ± 0.00	60.99 ± 0.21	4.44
MaskGAE	87.31 ± 0.05	75.20 ± 0.07	86.56 ± 0.26	90.52 ± 0.04	93.33 ± 0.14	92.31 ± 0.05	95.79 ± 0.02	70.99 ± 0.12	61.23 ± 0.19	3.00
S2GAE	86.15 ± 0.25	74.60 ± 0.06	$\mathbf{86.91 \pm 0.28}$	$\mathbf{90.94 \pm 0.08}$	$\mathbf{93.61 \pm 0.10}$	91.70 ± 0.08	95.82 ± 0.03	$\mathbf{72.02 \pm 0.05}$	$\mathbf{63.33 \pm 0.12}$	**2.22**

5.3.2 Link Prediction

Datasets. For link prediction task, we include 3 Planetoid benchmarks (Cora, CiteSeer, and PubMed [149]), 3 challenging OGB benchmakrs [171] (ogbl-ddi, ogbl-collab, ogbl-pppa), and 2 popular social networks [172] (BlogCatelog and Flickr) for experiments. To have a fair comparison, we follow previous studies [94, 121] to construct the train/valid/test edge sets, and evaluate model performance based on Hit rate (Hits@N) for OGB datasets, while AUC and AP (Average Precision) scores for other datasets.

Competitors. We compare our S2GAE model with 7 generative SSL methods: GAE [94], GraphSAGE [22], ARGVA [103], GPT-GNN [107], RRL [166], GraphMAE [6] and MaskGAE [170],

and 4 SOTA contrastive methods: DGI [51], GIC [96], MVGRL [50], and BGRL [1]. Since four contrastive methods, GPT-GNN, and GraphMAE are not officially tested on link prediction tasks, we apply them for this task by training an MLP-based decoder (a.k.a. finetune).

Table 5.6: Link prediction results on Planetoid benchmarks and social networks. The best results are highlighted.

	Cora AP	CiteSeer AP	PubMed AP	Blog. AP	Flickr AP	ogbl-ddi Hits@30	ogbl-collab Hits@100	ogbl-ppa Hits@50
DGI	90.61 ± 1.00	95.72 ± 0.10	92.23 ± 0.50	73.81 ± 1.72	92.38 ± 0.22	–	–	–
GIC	93.33 ± 0.70	$\mathbf{96.80 \pm 0.50}$	93.54 ± 0.30	61.96 ± 2.55	86.08 ± 0.16	–	–	–
MVGRL	87.78 ± 0.51	89.90 ± 0.42	87.69 ± 0.38	70.53 ± 1.98	91.32 ± 0.29	–	–	–
BGRL	86.59 ± 0.18	79.02 ± 0.22	95.56 ± 0.13	70.28 ± 1.77	88.58 ± 0.21	–	–	–
GAE	92.83 ± 0.03	91.68 ± 0.05	96.50 ± 0.02	84.87 ± 1.57	92.43 ± 0.26	51.56 ± 4.19	52.30 ± 1.01	$\mathbf{10.82 \pm 1.04}$
GraphSage	88.24 ± 0.87	87.90 ± 2.54	89.44 ± 0.82	77.09 ± 0.87	89.30 ± 0.16	65.80 ± 6.94	60.23 ± 1.20	8.92 ± 2.28
ARGE	93.23 ± 0.00	93.03 ± 0.00	97.11 ± 0.00	72.29 ± 0.36	87.05 ± 0.08	23.86 ± 6.53	37.66 ± 1.98	3.83 ± 0.84
GPT-GNN	92.53 ± 0.63	92.09 ± 0.76	97.79 ± 0.04	85.00 ± 0.29	93.05 ± 0.14	42.57 ± 6.01	51.32 ± 1.45	7.88 ± 0.92
RRL	89.60 ± 1.45	87.84 ± 1.18	91.47 ± 0.57	82.22 ± 0.86	92.76 ± 0.05	19.33 ± 2.51	40.52 ± 2.23	3.85 ± 0.83
GraphMAE	90.32 ± 0.01	92.66 ± 0.35	94.07 ± 0.02	77.54 ± 1.06	88.94 ± 0.02	–	33.62 ± 1.50	3.62 ± 0.91
MaskGAE	$\mathbf{96.29 \pm 0.23}$	$\mathbf{98.25 \pm 0.16}$	$\mathbf{98.99 \pm 0.06}$	80.12 ± 2.79	$\underline{92.87 \pm 0.36}$	19.27 ± 1.59	39.98 ± 0.74	3.77 ± 0.06
S2GAE-SAGE	94.50 ± 0.86	94.68 ± 0.34	97.11 ± 0.19	78.99 ± 1.91	91.71 ± 0.20	$\mathbf{75.18 \pm 6.57}$	55.44 ± 0.82	4.79 ± 0.16
S2GAE-GCN	94.46 ± 0.24	93.81 ± 0.40	98.22 ± 0.05	$\mathbf{86.12 \pm 0.49}$	$\mathbf{94.10 \pm 0.04}$	$\mathbf{75.02 \pm 2.26}$	$\mathbf{61.01 \pm 1.18}$	9.97 ± 1.55

Results. Tables 5.4 and 5.6 list the results on all evaluation metrics. We found that S2GAE outperforms both generative and contrastive baselines on 5 of 8 datasets and achieves comparable results on other 2 of 3 datasets, with substantial performance gains on challenging OGB datasets. Moreover, the generative baselines (i.e., GAE variants) perform generally better than contrastive methods across 8 datasets, which is consistent with common belief. Compared with MaskGAE, S2GAE wins on 5 large-scale datasets, including OGB benchmarks, while losing to 3 small datasets. This result sheds light on the scalability of our S2GAE model.

5.3.3 Node Classification

Datasets. For node classification task, we consider 4 widely-used datasets [165] (Amazon-Computers, Amazon-Photo, Coauthor-CS, and Coauthor-Physics), 3 Planetoid graphs (Cora, CiteSeer, and PubMed), and 2 large-scale OGB datasets [121], ogbn-arxiv and ogbn-proteins. Following previous studies [1, 121], we evaluate the model performance based on AUC and accuracy

Table 5.7: Graph classification performance on both contrastive and generative methods. A.R. is the average rank.

	Dataset	IMDB-B	IMDB-M	PROTEINS	COLLAB	MUTAG	REDDIT-B	NCI1	A.R.
Graph Kernels	WL	72.30 ± 3.44	46.95 ± 0.46	72.92 ± 0.56	–	80.72 ± 3.00	68.82 ± 0.41	80.31 ± 0.46	8.00
	DGK	66.96 ± 0.56	44.55 ± 0.52	73.30 ± 0.82	–	87.44 ± 2.72	78.04 ± 0.39	80.31 ± 0.46	7.66
Unsupervised	graph2vec	71.10 ± 0.54	50.44 ± 0.87	73.30 ± 2.05	–	83.15 ± 9.25	75.78 ± 1.03	73.22 ± 1.81	7.83
	Infograph	73.03 ± 0.87	49.69 ± 0.53	74.44 ± 0.31	70.65 ± 1.13	89.01 ± 1.13	82.50 ± 1.42	76.20 ± 1.06	5.57
Contrastive	GraphCL	71.14 ± 0.44	48.58 ± 0.67	74.39 ± 0.45	71.36 ± 1.15	86.80 ± 1.34	$\underline{89.53 \pm 0.84}$	77.87 ± 0.41	6.28
	JOAO	70.21 ± 3.08	49.20 ± 0.77	74.55 ± 0.41	69.50 ± 0.36	87.35 ± 1.02	85.29 ± 1.35	78.07 ± 0.47	6.57
	GCC	72.0	49.4	–	78.9	–	$\mathbf{89.8}$	–	4.75
	MVGRL	74.20 ± 0.70	51.20 ± 0.50	–	–	$\underline{89.70 \pm 1.10}$	84.50 ± 0.60	–	4.00
	InfoGCL	75.10 ± 0.90	51.40 ± 0.80	–	80.00 ± 1.30	$\mathbf{91.20 \pm 1.30}$	–	80.20 ± 0.60	3.00
Generative	GraphMAE	$\underline{75.52 \pm 0.66}$	$\underline{51.63 \pm 0.52}$	$\underline{75.30 \pm 0.39}$	$\underline{80.32 \pm 0.46}$	88.19 ± 1.26	88.01 ± 0.19	$\underline{80.40 \pm 0.30}$	2.57
	S2GAE	$\mathbf{75.76 \pm 0.62}$	$\mathbf{51.79 \pm 0.36}$	$\mathbf{76.37 \pm 0.43}$	$\mathbf{81.02 \pm 0.53}$	88.26 ± 0.76	87.83 ± 0.27	$\mathbf{80.80 + 0.24}$	**1.85**

(ACC) scores for ogbn-proteins and other datasets, respectively.

Competitors. To have a rigorous and fair comparison between generative and contrastive SSL methods on graphs, we include 7 SOTA generative SSL baselines: GAE [94], GraphSAGE [22], ARGVA [103], GPT-GNN [107], RRL [166], GraphMAE [6] and MaskGAE [170], and 4 SOTA contrastive arts: DGI [51], GIC [96], MVGRL [50], and BGRL [1]. After the model is trained, we use the whole dataset to generate node representations for downstream evaluation. Then, we train a LIBSVM classifier on the learned node presentations of all models, and apply 5-fold cross-validation to estimate the performance. To avoid randomness, we repeat the process 10 times and report the averaged results.

Results. Tables 5.5 reports the results. We found that S2GAE outperforms all baselines on 5 of 9 datasets and especially performs well on two large-scale OGB benchmarks. Compared with traditional generative efforts (excluding GraphMAE, MaskGAE, and S2GA2), contrastive methods achieve SOTA results on almost all classification scenarios. However, by refining GAE from the masked autoencoding aspect, S2GAE, GraphMAE, and MaskGAE tend to perform on par with or even better than SOTA contrastive methods. Specifically, our S2GAE performs especially well on relatively large and more challenging benchmarks like orgn-arxiv and ogbn-proteins, compared with GraphMAE and MaskGAE. These results validate the generalization ability of S2GAE on node classification.

5.3.4 Graph Classification

Datasets. For graph classification task, we test on 7 widely used benchmark datasets [156]: MUTAG, IMDB-B, IMDB-M, PROTEINS, COLLAB, REDDIT-B, and NCI1, where each of them is a collection of graphs. Following [6], we use node labels in MUTAG, PROTEINS, and NCI1 as features, whereas node degrees in IMDB-B, IMDB-M, REDDIT-B, and COLLAB are used. We feed the encoded graph-level representation into a downstream LIBSVM classifier to predict the label, and report the mean 10-fold cross-validation accuracy with standard deviation after 5 runs.

Competitors. Follow previous studies [6, 48], we compare our S2GAE model with 2 graph kernel methods: Weisfeiler-Lehman sub-tree kernel–WL [155] and deep graph kernel–DGK [156], 7 SOTA unsupervised and contrastive methods: graph2vec [158], Infograph [137], GraphCL [48], JOAO [53], GCC [49], MVGRL [50], and InfoGCL [99], and 1 generative method–GraphMAE [6]. If available, we report results from previous papers and adopt GIN [135] as our encoder. We don't include MaskGAE for comparison because its source code does not support graph classification.

Results. Table 5.7 summarizes the results. We observed that our S2GAE outperforms existing contrastive and generative baselines on 5 of 7 datasets. In these benchmarks, only graph-level contrastive models have been demonstrated to be effective. These results substantiate the generalization ability of S2GAE on graph-level tasks.

To summarize, our S2GAE not only inherits GAE's capacity for link prediction (See Table 5.4 and 5.6), but also generalizes well to node-level (in Table 5.5) and graph-level (in Table 5.7) classification tasks, which was previously dominated by contrastive methods. The results manifest the potential of GAE on graphs as a universal representation learner for a variety of applications.

5.3.5 Ablation Study

We further conduct a series of ablation studies to verify the effectiveness of our design choices. Due to limited space, we selected 4 OGB benchmarks across link prediction and node classification for this study. We follow the same experimental setups as above.

Self-supervised training of S2GAE is crucial. The core idea of S2GAE is to train GAE under

Table 5.8: Ablation studies of S2GAE. "GP" indicates graph perturbation. "MGR" means masked graph reconstruction. "GCN" means GNN encoder similar to [6]. "Concat" means concatenate all intermediate representations.

		Link Prediction		Node Classification	
		ddi	collab	arxiv	proteins
COM.	S2GAE	65.91 ± 3.50	54.74 ± 1.06	72.02 ± 0.05	63.33 ± 0.12
	w/o GP	44.12 ± 2.42	51.88 ± 0.57	70.28 ± 0.07	61.74 ± 0.23
	w/o MGR	54.27 ± 4.78	50.29 ± 1.26	70.70 ± 0.04	60.46 ± 0.15
Decoder	Concat	57.77 ± 3.30	48.26 ± 1.26	70.82 ± 0.02	60.15 ± 0.20
	GCN	53.50 ± 5.16	45.75 ± 1.52	71.00 ± 0.01	59.82 ± 0.22
	w/o CC	24.98 ± 5.19	42.20 ± 1.68	68.40 ± 0.12	57.69 ± 0.13

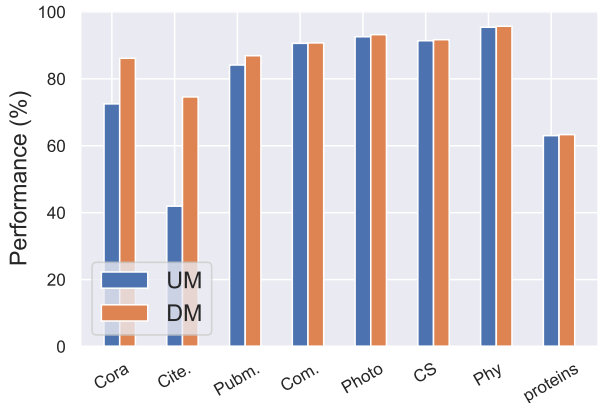

Figure 5.3: Ablation study of graph masking strategies on node classification.

a self-supervised setting. To verify the necessity, we explore the contributions of graph masking (w/o GP) and masked edge reconstruction (w/o MGR) in Table 5.8. In general, by considering them jointly, S2GAE outperforms either of them by a great margin. For example, S2GAE improves 21.44% and 1.86% of the best of them on ddi and arxiv, respectively. This result validates our motivation to improve GAE from an SSL perspective.

Effect of cross-correlation decoder. Capturing the cross-correlation between nodes is important to predict masked edges. To verify this, we compare S2GAE with different decoder variants in Table 5.8. First, the performance gap between S2GAE and the "Concat" (or "GCN") variant

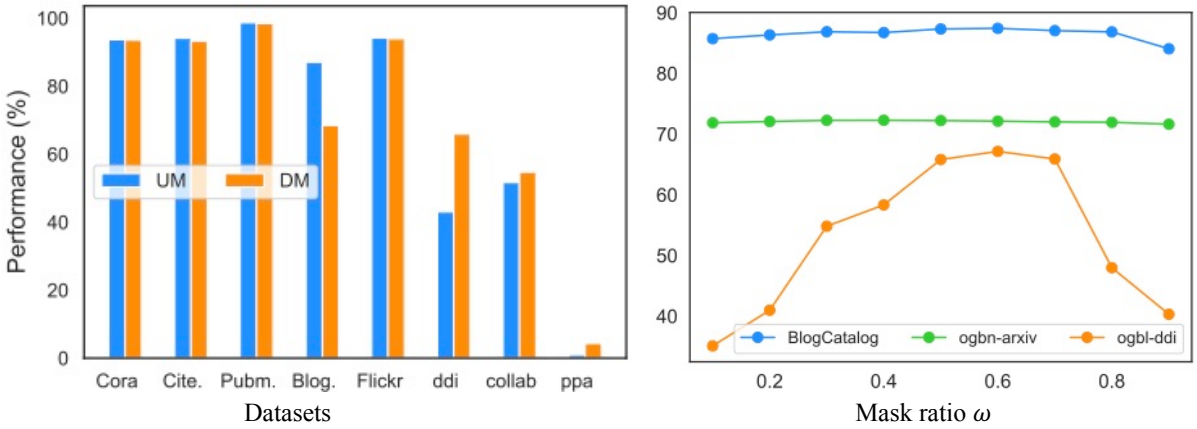

Figure 5.4: Ablation study of graph masking strategies (left panel) and masking ratio (right panel).

shows that concatenating all hidden representations (or adopting an advanced GNN model as decoder) does not help to reconstruct masked edges from noisy representations. Second, without the cross-correlation decoder, the resultant variant (w/o CC) performs poorly on 4 evaluation scenarios. These observations demonstrate the effectiveness of the proposed cross-correlation decoder.

Effect of masking ratio. The right panel of Figure 5.4 shows the influences of masking ratios. In general, a small marking ratio (e.g., $\omega < 0.5$) is not good enough to learn generalizable features, and our S2GAE model performs consistently well when ω reaches 0.7. Similar tendencies are observed on other datasets, and the detailed configurations are listed in the original paper.

Effect of direction-aware graph masking. The left panel of Figure 5.4 shows the influence of graph masking strategies. We observed that DM is more effective in the OGB datasets, which are more sparse than the others. These results verify the necessary to perform direction-aware graph masking for different types of graphs. We also see that DM generally performs better on node classification tasks (See Table 5.3).

6. CONCLUSION AND FUTURE WORK

6.1 Conclusion

In this thesis, we present a series of efficient algorithms aimed at enhancing self-supervised learning on graphs at various levels. Our contributions encompass a personalized approach to accelerate augmentation selection, the elimination of graph augmentation through model perturbation, the distillation of fine-tuned GNN models into lightweight neural networks using reliable knowledge distillation, and the development of universal graph models to improve their applicability.

First, we extend the traditional architecture search technique to address the local augmentation selection problem in graph contrastive learning. We advance conventional GCL by enabling each graph to choose its own suitable augmentation operations. To handle the vast search space, we design a tailored augmentation selector that converts the discrete space into a continuous one. This module can be effectively trained with downstream GCL models in an end-to-end fashion. Through visualization of the learned augmentation distributions across different datasets, we demonstrate that our approach can effectively identify the most appropriate augmentations for each graph based on its characteristics. Furthermore, as an augmentation selection method, our model has broad applicability. It can flexibly incorporate new augmentation strategies created by domain experts by modifying the augmentation pool without significant additional effort.

Next, to further accelerate the training efficiency of existing self-supervised graph models, we propose an alternative contrastive learning framework based on model perturbation. Instead of searching for the optimal combination of perturbing nodes, edges, or attributes, we conduct perturbations on the model architecture itself, specifically the GNNs. Model perturbation represents a global disturbance of all nodes in the graph and is thus more efficient than data augmentation. To enhance the effectiveness of our proposal, we introduce two customized perturbation strategies for GNNs: "randMP" and "weightPrune." The former injects randomness into the message propa-

gation of GNNs to increase local variances between opposing views, while the latter sparsifies the weight transformation of GNNs by pruning weights to enhance the diversity of views.

However, accelerating the pre-training of GNN models does not directly improve their efficiency for inference. The major hindrance lies in the heavy data dependency incurred by the message propagation among multi-hop neighbors in GNNs, resulting in substantial computational costs and memory footprints. To address this issue, we propose compressing the pre-trained GNN model into a lightweight MLP student through noise-aware distillation. Specifically, we employ a meta-policy to filter out unreliable soft labels by determining whether each node's representations should be used in distillation. The student model then only distills the soft labels of the nodes retained by the meta-policy. To train the meta-policy, we design a reward-driven objective based on a meta-set, where the meta-policy is rewarded for making accurate filtering decisions. The meta-policy is optimized using policy gradient to achieve the highest expected reward and is subsequently applied to unlabeled nodes. Through iterative updates to the meta-policy and the student model, we achieve a win-win scenario: significantly improved performance of the vanilla MLP student by providing reliable guidance while maintaining the inference efficiency of MLPs without increasing the model size.

Finally, we explore methods to further enhance the generalization capability of self-supervised GNN models, enabling a single model to be effectively applied to various graph analysis tasks, thus reducing deployment efforts. To achieve this goal, we harness the power of the standard graph autoencoder framework with minimal yet impactful modifications. Instead of reconstructing the entire input structure, we randomly mask a portion of edges and learn to reconstruct these missing edges using an effective masking strategy and an expressive decoder network. Moreover, we theoretically demonstrate the effectiveness of our approach by revealing that it can be regarded as an edge-level contrastive learning framework, providing insights into its ability to generalize well to both link prediction and classification tasks.

6.2 Future Work

My lifelong goal is to conduct impactful research that can be applied to industries, creating tangible value for society. To achieve this, I plan to maintain an adaptive research agenda that keeps me updated with emerging challenges in the long run. In the short term, my focus is on addressing the remaining challenges in my current research directions while exploring new frontiers. Specifically, I have outlined my research plan in three directions:

Tiny machine learning. The research landscape, not only for graph models, has shifted from traditional high-end systems to low-end clients such as edge computing and Internet of Things (IoT) devices. This transition necessitates efficient machine learning approaches due to memory and latency constraints. Our model RKD-MLP [173] serves as a preliminary work on graphs. However, it is still in its early stages, as we observe a noticeable performance gap when dealing with large-scale datasets. Additionally, designing compact graph models for tasks beyond classification, such as link prediction, remains unclear. Consequently, I will continue developing scalable and cost-efficient graph models.

Few-shot/zero-shot learning. The data hunger problem poses a significant challenge when applying deep models in industries. Developing a powerful graph model for a specific downstream task often requires a substantial amount of annotated samples, whereas limited labeled data is common in real-world scenarios. I have recently explored self-supervised learning [174, 18] as a means to alleviate the demand for labeled data to some extent. However, achieving satisfactory classification results still requires a certain number of labeled samples for fine-tuning. It is crucial to explore more labeling-efficient techniques to make models applicable in the industry.

Trustworthy ecosystems. In addition to advancing model performance and reducing latency, deploying deep models in practice raises serious concerns related to ethics, fairness, safety, and more. For instance, it has been revealed that state-of-the-art deep models may exhibit different behavior across individuals based on characteristics such as race, gender, disabilities, and sexual or political orientation. Moreover, the security of deep models can be easily compromised by adversarial samples that introduce imperceptible perturbations to normal data. These limitations

significantly restrict the widespread adoption of deep models as universal solutions in high-stakes applications like financial systems and self-driving cars, as even a minor mistake can lead to substantial losses and potentially endanger lives. My prior work on model interpretability [55, 175] aims to improve model transparency. To further pursue this direction, I plan to study adversarial attacks and defenses for deep models on graphs, as well as explore model fairness to achieve a fair and robust machine learning system.

Interdisciplinary research. I have valuable interdisciplinary collaboration experiences with scientists and domain experts from both academia and industry. These collaborations have not only broadened my horizons but also amplified the impact of my research. For example, I am currently working on a healthcare project with researchers at UTHealth Houston, focusing on applying machine learning to improve the fairness and predictive performance of organ transplantation [176] and clinical trial matching [177]. Additionally, I have collaborated with engineers from Alibaba DAOMO Academy to develop effective recommendation engines for online shopping websites and with Samsung Research America to enhance real-time bidding systems through model compression. These interdisciplinary endeavors have inspired me to leverage my expertise in machine learning to drive positive societal outcomes. As a long-term goal, I aspire to pursue interdisciplinary research opportunities across various domains such as social science, security, biochemistry, and healthcare.

REFERENCES

[1] S. Thakoor, C. Tallec, M. G. Azar, M. Azabou, E. L. Dyer, R. Munos, P. Veličković, and M. Valko, "Large-scale representation learning on graphs via bootstrapping," in *International Conference on Learning Representations*, 2022.

[2] J. Xia, L. Wu, J. Chen, B. Hu, and S. Z. Li, "Simgrace: A simple framework for graph contrastive learning without data augmentation," in *International World Wide Web Conference*, pp. 1070–1079, 2022.

[3] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang, "Deep graph contrastive representation learning," in *ICML Workshop on Graph Representation Learning and Beyond*, 2020.

[4] H. Zhang, Q. Wu, J. Yan, D. Wipf, and P. S. Yu, "From canonical correlation analysis to self-supervised graph neural networks," in *Advances in Neural Information Processing Systems*, vol. 34, pp. 76–89, 2021.

[5] S. Zhang, Y. Liu, Y. Sun, and N. Shah, "Graph-less neural networks: Teaching old mlps new tricks via distillation," in *International Conference on Learning Representations*, 2022.

[6] Z. Hou, X. Liu, Y. Cen, Y. Dong, H. Yang, C. Wang, and J. Tang, "Graphmae: Self-supervised masked graph autoencoders," in *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 594–604, 2022.

[7] Z. Zhou, S. Zhou, B. Mao, X. Zhou, J. Chen, Q. Tan, D. Zha, C. Wang, Y. Feng, and C. Chen, "Opengsl: benchmarking graph structure learning," in *arXiv preprint arXiv:2306.10280*, 2023.

[8] Q. Tan, X. Zhang, N. Liu, D. Zha, L. Li, R. Chen, S.-H. Choi, and X. Hu, "Bring your own view: Graph neural networks for link prediction with personalized subgraph selection," in *ACM International Conference on Web Search and Data Mining*, pp. 625–633, 2023.

[9] S. Zhou, X. Huang, N. Liu, Q. Tan, and F.-L. Chung, "Unseen anomaly detection on networks via multi-hypersphere learning," in *SIAM International Conference on Data Mining*, pp. 262–270, SIAM, 2022.

[10] J. Dong, Q. Zhang, X. Huang, Q. Tan, D. Zha, and Z. Zihao, "Active ensemble learning for knowledge graph error detection," in *ACM International Conference on Web Search and Data Mining*, pp. 877–885, 2023.

[11] J. Dong, Q. Zhang, X. Huang, K. Duan, Q. Tan, and Z. Jiang, "Hierarchy-aware multi-hop question answering over knowledge graphs," in *International World Wide Web Conference*, pp. 2519–2527, 2023.

[12] H. Zhou, S. Zhou, K. Duan, X. Huang, Q. Tan, and Z. Yu, "Interest driven graph structure learning for session-based recommendation," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 284–296, Springer, 2023.

[13] D. Zha, L. Feng, Q. Tan, Z. Liu, K.-H. Lai, B. Bhushanam, Y. Tian, A. Kejariwal, and X. Hu, "Dreamshard: Generalizable embedding table placement for recommender systems," in *Advances in Neural Information Processing Systems*, 2022.

[14] H. Zhou, Q. Tan, X. Huang, K. Zhou, and X. Wang, "Temporal augmented graph neural networks for session-based recommendations," in *International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1798–1802, 2021.

[15] Q. Tan, J. Zhang, N. Liu, X. Huang, H. Yang, J. Zhou, and X. Hu, "Dynamic memory based attention network for sequential recommendation," in *AAAI Conference on Artificial Intelligence*, vol. 35, pp. 4384–4392, 2021.

[16] Z. Jiang, K. Zhou, M. Zhang, R. Chen, X. Hu, and S.-H. Choi, "Adaptive risk-aware bidding with budget constraint in display advertising," *arXiv preprint arXiv:2212.12533*, 2022.

[17] D. Zha, L. Feng, L. Luo, B. Bhushanam, Z. Liu, Y. Hu, J. Nie, Y. Huang, Y. Tian, A. Kejariwal, and X. Hu, "Pre-train and search: Efficient embedding table sharding with pre-trained neural cost models," in *Conference on Machine Learning and Systems*, 2023.

[18] Q. Tan, S. Ding, N. Liu, S.-H. Choi, L. Li, R. Chen, and X. Hu, "Graph contrastive learning with model perturbation," 2022.

[19] X. Han, Z. Jiang, N. Liu, and X. Hu, "G-mixup: Graph data augmentation for graph classification," in *International Conference on Machine Learning*, pp. 1–9, PMLR, 2022.

[20] X. Han, Z. Jiang, N. Liu, Q. Song, J. Li, and X. Hu, "Geometric graph representation learning via maximizing rate reduction," in *Proceedings of the ACM Web Conference 2022*, pp. 1226–1237, 2022.

[21] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations*, 2017.

[22] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[23] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*, 2017.

[24] Q. Tan, X. Zhang, X. Huang, H. Chen, J. Li, and X. Hu, "Collaborative graph neural networks for attributed network embedding," *IEEE Transactions on Knowledge and Data Engineering*, 2023.

[25] Q. Tan, G. Yu, J. Wang, C. Domeniconi, and X. Zhang, "Individuality-and commonality-based multiview multilabel learning," *IEEE Transactions on Cybernetics*, vol. 51, no. 3, pp. 1716–1727, 2019.

[26] Q. Tan, G. Yu, C. Domeniconi, J. Wang, and Z. Zhang, "Incomplete multi-view weak-label learning.," in *International Joint Conferences on Artificial Intelligence*, pp. 2703–2709, 2018.

[27] D. Zha, Z. P. Bhat, K.-H. Lai, F. Yang, Z. Jiang, S. Zhong, and X. Hu, "Data-centric artificial intelligence: A survey," *arXiv preprint arXiv:2303.10158*, 2023.

[28] Q. Tan, G. Yu, C. Domeniconi, J. Wang, and Z. Zhang, "Multi-view weak-label learning based on matrix completion," in *SIAM International Conference on Data Mining*, pp. 450–458, SIAM, 2018.

[29] Q. Tan, Y. Yu, G. Yu, and J. Wang, "Semi-supervised multi-label classification using incomplete label information," *Neurocomputing*, vol. 260, pp. 192–202, 2017.

[30] Q. Tan, Y. Liu, X. Chen, and G. Yu, "Multi-label classification based on low rank representation for image annotation," *Remote Sensing*, vol. 9, no. 2, p. 109, 2017.

[31] D. Zha, K.-H. Lai, Q. Tan, S. Ding, N. Zou, and X. B. Hu, "Towards automated imbalanced learning with deep hierarchical reinforcement learning," in *ACM International Conference on Information & Knowledge Management*, pp. 2476–2485, 2022.

[32] Z. Jiang, X. Han, C. Fan, F. Yang, A. Mostafavi, and X. Hu, "Generalized demographic parity for group fairness," in *International Conference on Learning Representations*, 2022.

[33] Z. Jiang, X. Han, H. Jin, G. Wang, N. Zou, and X. Hu, "Weight perturbation can help fairness under distribution shift," *arXiv preprint arXiv:2303.03300*, 2023.

[34] Z. Jiang, K. Zhou, Z. Liu, L. Li, R. Chen, S.-H. Choi, and X. Hu, "An information fusion approach to learning with instance-dependent label noise," in *International Conference on Learning Representations*, 2022.

[35] Y. Yang and Z. Xu, "Rethinking the value of labels for improving class-imbalanced learning," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 19290–19301, 2020.

[36] S. Zhou, Q. Tan, Z. Xu, X. Huang, and F.-l. Chung, "Subtractive aggregation for attributed network anomaly detection," in *ACM International Conference on Information & Knowledge Management*, pp. 3672–3676, 2021.

[37] Y. Xie, Z. Xu, J. Zhang, Z. Wang, and S. Ji, "Self-supervised learning of graph neural networks: A unified review," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[38] Y. Liu, M. Jin, S. Pan, C. Zhou, Y. Zheng, F. Xia, and P. Yu, "Graph self-supervised learning: A survey," *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[39] Q. Zhang, J. Dong, Q. Tan, and X. Huang, "Integrating entity attributes for error-aware knowledge graph embedding," *IEEE Transactions on Knowledge and Data Engineering*, 2023.

[40] D. Zha, K.-H. Lai, K. Zhou, and X. Hu, "Simplifying deep reinforcement learning via self-supervision," *arXiv preprint arXiv:2106.05526*, 2021.

[41] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, "Context encoders: Feature learning by inpainting," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2536–2544, 2016.

[42] R. Zhang, P. Isola, and A. A. Efros, "Colorful image colorization," in *European Conference on Computer Vision*, pp. 649–666, Springer, 2016.

[43] M. Noroozi and P. Favaro, "Unsupervised learning of visual representations by solving jigsaw puzzles," in *European Conference on Computer Vision*, pp. 69–84, Springer, 2016.

[44] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[45] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9729–9738, 2020.

[46] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International Conference on Machine Learning*, pp. 1597–1607, PMLR, 2020.

[47] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar, *et al.*, "Bootstrap your own latent-a new approach to self-supervised learning," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 21271–21284, 2020.

[48] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, "Graph contrastive learning with augmentations," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 5812–5823, 2020.

[49] J. Qiu, Q. Chen, Y. Dong, J. Zhang, H. Yang, M. Ding, K. Wang, and J. Tang, "Gcc: Graph contrastive coding for graph neural network pre-training," in *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1150–1160, 2020.

[50] K. Hassani and A. H. Khasahmadi, "Contrastive multi-view representation learning on graphs," in *International Conference on Machine Learning*, pp. 4116–4126, PMLR, 2020.

[51] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," in *International Conference on Learning Representations*, 2019.

[52] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang, "Graph contrastive learning with adaptive augmentation," in *International World Wide Web Conference*, pp. 2069–2080, 2021.

[53] Y. You, T. Chen, Y. Shen, and Z. Wang, "Graph contrastive learning automated," in *International Conference on Machine Learning*, 2021.

[54] D. B. West *et al.*, *Introduction to graph theory*, vol. 2. Prentice hall Upper Saddle River, 2001.

[55] N. Liu, Q. Tan, Y. Li, H. Yang, J. Zhou, and X. Hu, "Is a single vector enough? exploring node polysemy for network embedding," in *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 932–940, 2019.

[56] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann, "Tudataset: A collection of benchmark datasets for learning with graphs," in *ICML 2020 Workshop on Graph Representation Learning and Beyond*, 2020.

[57] Y. Yin, Q. Wang, S. Huang, H. Xiong, and X. Zhang, "Autogcl: Automated graph contrastive learning via learnable view generators," in *AAAI Conference on Artificial Intelligence*, vol. 36, pp. 8892–8900, 2022.

[58] S. Suresh, P. Li, C. Hao, and J. Neville, "Adversarial graph augmentation to improve graph contrastive learning," in *Advances in Neural Information Processing Systems*, vol. 34, pp. 15920–15933, 2021.

[59] N. Lee, J. Lee, and C. Park, "Augmentation-free self-supervised learning on graphs," in *AAAI Conference on Artificial Intelligence*, vol. 36, pp. 7372–7380, 2022.

[60] L. Wu, H. Lin, C. Tan, Z. Gao, and S. Z. Li, "Self-supervised learning on graphs: Contrastive, generative, or predictive," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[61] T. Chen, Y. Sui, X. Chen, A. Zhang, and Z. Wang, "A unified lottery ticket hypothesis for graph neural networks," in *International Conference on Machine Learning*, pp. 1695–1706, PMLR, 2021.

[62] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," in *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 257–266, 2019.

[63] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "GraphSAINT: Graph sampling based inductive learning method," in *International Conference on Learning Representations*, 2020.

[64] X. Liu, M. Yan, L. Deng, G. Li, X. Ye, D. Fan, S. Pan, and Y. Xie, "Survey on graph neural network acceleration: An algorithmic perspective," *International Joint Conferences on Artificial Intelligence*, 2022.

[65] H. Zhou, A. Srivastava, H. Zeng, R. Kannan, and V. Prasanna, "Accelerating large scale real-time gnn inference using channel pruning," *Proceedings of the VLDB Endowment*, vol. 14, no. 9, pp. 1597–1605, 2021.

[66] S. A. Tailor, J. Fernandez-Marques, and N. D. Lane, "Degree-quant: Quantization-aware training for graph neural networks," *International Conference on Learning Representations*, 2021.

[67] G. Hinton, O. Vinyals, J. Dean, *et al.*, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, vol. 2, no. 7, 2015.

[68] V. G. Satorras and J. B. Estrach, "Few-shot learning with graph neural networks," in *International Conference on Learning Representations*, 2018.

[69] W. Feng, J. Zhang, Y. Dong, Y. Han, H. Luan, Q. Xu, Q. Yang, E. Kharlamov, and J. Tang, "Graph random neural networks for semi-supervised learning on graphs," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 22092–22103, 2020.

[70] J. Gasteiger, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," in *International Conference on Learning Representations*, 2018.

[71] G. Wang, R. Ying, J. Huang, and J. Leskovec, "Multi-hop attention graph neural network," in *International Joint Conferences on Artificial Intelligence*, 2021.

[72] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, "Masked autoencoders are scalable vision learners," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16000–16009, 2022.

[73] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, *et al.*, "Segment anything," *arXiv preprint arXiv:2304.02643*, 2023.

[74] C. Wei, H. Fan, S. Xie, C.-Y. Wu, A. Yuille, and C. Feichtenhofer, "Masked feature prediction for self-supervised visual pre-training," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14668–14678, 2022.

[75] S. Zheng, J. Lu, H. Zhao, X. Zhu, Z. Luo, Y. Wang, Y. Fu, J. Feng, T. Xiang, P. H. Torr, *et al.*, "Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6881–6890, 2021.

[76] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin, "Graph neural networks for social recommendation," in *The World Wide Web Conference*, pp. 417–426, 2019.

[77] R. Hu, S. Pan, G. Long, Q. Lu, L. Zhu, and J. Jiang, "Going deep: Graph convolutional ladder-shape networks," in *AAAI Conference on Artificial Intelligence*, vol. 34, pp. 2838–2845, 2020.

[78] L. Cai, Z. Chen, C. Luo, J. Gui, J. Ni, D. Li, and H. Chen, "Structural temporal graph neural networks for anomaly detection in dynamic graphs," in *ACM International Conference on Information & Knowledge Management*, pp. 3747–3756, 2021.

[79] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2020.

[80] L. Wang, W. Yu, W. Wang, W. Cheng, W. Zhang, H. Zha, X. He, and H. Chen, "Learning robust representations with graph denoising policy network," in *IEEE International Conference on Data Mining*, pp. 1378–1383, IEEE, 2019.

[81] R. G. Iyer, W. Wang, and Y. Sun, "Bi-level attention graph neural networks," in *IEEE International Conference on Data Mining*, pp. 1126–1131, IEEE, 2021.

[82] H. Wang, J. Zhang, Q. Zhu, and W. Huang, "Augmentation-free graph contrastive learning," *arXiv preprint arXiv:2204.04874*, 2022.

[83] Q. Tan, N. Liu, and X. Hu, "Deep representation learning for social network analysis," *Frontiers in Big Data*, vol. 2, p. 2, 2019.

[84] H. Li, X. Wang, Z. Zhang, and W. Zhu, "Ood-gnn: Out-of-distribution generalized graph neural network," *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[85] Z. Jiang, X. Han, C. Fan, Z. Liu, N. Zou, A. Mostafavi, and X. Hu, "Fmp: Toward fair graph message passing against topology bias," *arXiv preprint arXiv:2202.04187*, 2022.

[86] H. Ling, Z. Jiang, Y. Luo, S. Ji, and N. Zou, "Learning fair graph representations via automated data augmentations," in *International Conference on Learning Representations*, 2023.

[87] Z. Jiang, X. Han, C. Fan, Z. Liu, X. Huang, N. Zou, A. Mostafavi, and X. Hu, "Topology matters in fair graph learning: a theoretical pilot study," 2023.

[88] Z. Jiang, X. Han, C. Fan, Z. Liu, N. Zou, A. Mostafavi, and X. Hu, "Fair graph message passing with transparency," 2023.

[89] H. Ling, Z. Jiang, M. Liu, S. Ji, and N. Zou, "Graph mixup with soft alignments," in *International Conference on Machine Learning*, PMLR, 2023.

[90] Z. Liu, Z. Jiang, S. Zhong, K. Zhou, L. Li, R. Chen, S.-H. Choi, and X. Hu, "Editable graph neural network for node classifications," *arXiv preprint arXiv:2305.15529*, 2023.

[91] J. Chen, T. Ma, and C. Xiao, "Fastgcn: Fast learning with graph convolutional networks via importance sampling," in *International Conference on Learning Representations*, 2018.

[92] M. Ding, J. Tang, and J. Zhang, "Semi-supervised learning on graphs with generative adversarial nets," in *ACM International Conference on Information and Knowledge Management*, pp. 913–922, 2018.

[93] K.-H. Lai, D. Zha, K. Zhou, and X. Hu, "Policy-gnn: Aggregation optimization for graph neural networks," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2020.

[94] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.

[95] A. Garcia Duran and M. Niepert, "Learning graph representations with embedding propagation," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[96] C. Mavromatis and G. Karypis, "Graph infoclust: Maximizing coarse-grain mutual information in graphs," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 541–553, Springer, 2021.

[97] X. Liu, F. Zhang, Z. Hou, L. Mian, Z. Wang, J. Zhang, and J. Tang, "Self-supervised learning: Generative or contrastive," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[98] P. Bielak, T. Kajdanowicz, and N. V. Chawla, "Graph barlow twins: A self-supervised representation learning framework for graphs," *arXiv preprint arXiv:2106.02466*, 2021.

[99] D. Xu, W. Cheng, D. Luo, H. Chen, and X. Zhang, "Infogcl: Information-aware graph contrastive learning," in *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[100] Y. Liu, S. Pan, M. Jin, C. Zhou, F. Xia, and P. S. Yu, "Graph self-supervised learning: A survey," *arXiv preprint arXiv:2103.00111*, 2021.

[101] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 701–710, 2014.

[102] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 855–864, 2016.

[103] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang, "Adversarially regularized graph autoencoder for graph embedding," in *International Joint Conference on Artificial Intelligence*, pp. 2609–2615, 2018.

[104] G. Cui, J. Zhou, C. Yang, and Z. Liu, "Adaptive graph encoder for attributed graph embedding," in *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 976–985, 2020.

[105] C. Wang, S. Pan, G. Long, X. Zhu, and J. Jiang, "Mgae: Marginalized graph autoencoder for graph clustering," in *ACM International Conference on Information and Knowledge Management*, pp. 889–898, 2017.

[106] J. Park, M. Lee, H. J. Chang, K. Lee, and J. Y. Choi, "Symmetric graph convolutional autoencoder for unsupervised graph representation learning," in *International Conference on Computer Vision*, pp. 6519–6528, 2019.

[107] Z. Hu, Y. Dong, K. Wang, K.-W. Chang, and Y. Sun, "Gpt-gnn: Generative pre-training of graph neural networks," in *ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 1857–1867, 2020.

[108] X. Zhang, Q. Tan, X. Huang, and B. Li, "Graph contrastive learning with personalized augmentation," *arXiv preprint arXiv:2209.06560*, 2022.

[109] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *International Conference on Machine Learning*, 2019.

[110] E. Rossi, F. Frasca, B. Chamberlain, D. Eynard, M. Bronstein, and F. Monti, "Sign: Scalable inception graph neural networks," *arXiv preprint arXiv:2004.11198*, 2020.

[111] C. Sun, H. Gu, and J. Hu, "Scalable and adaptive graph neural networks with self-label-enhanced training," *arXiv preprint arXiv:2104.09376*, 2021.

[112] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International Conference on Machine Learning*, pp. 1263–1272, PMLR, 2017.

[113] Y. Sui, X. Wang, T. Chen, X. He, and T.-S. Chua, "Inductive lottery ticket learning for graph neural networks," 2021.

[114] Y. Zhao, D. Wang, D. Bates, R. Mullins, M. Jamnik, and P. Lio, "Learned low precision graph neural networks," *arXiv preprint arXiv:2009.09232*, 2020.

[115] Y. Yang, J. Qiu, M. Song, D. Tao, and X. Wang, "Distilling knowledge from graph convolutional networks," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7074–7083, 2020.

[116] B. Yan, C. Wang, G. Guo, and Y. Lou, "Tinygnn: Learning efficient graph neural networks," in *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1848–1856, 2020.

[117] A. Deng and B. Hooi, "Graph neural network-based anomaly detection in multivariate time series," in *AAAI Conference on Artificial Intelligence*, vol. 35, pp. 4027–4035, 2021.

[118] Y. Xu, Y. Zhang, W. Guo, H. Guo, R. Tang, and M. Coates, "Graphsail: Graph structure aware incremental learning for recommender systems," in *ACM International Conference on Information and Knowledge Management*, pp. 2861–2868, 2020.

[119] Y. Zhuang, L. Lyu, C. Shi, C. Yang, and L. Sun, "Data-free adversarial knowledge distillation for graph neural networks," in *International Joint Conferences on Artificial Intelligence*, 2022.

[120] C. Yang, J. Liu, and C. Shi, "Extract the knowledge of graph neural networks and go beyond it: An effective knowledge distillation framework," in *International Conference on World Wide Web*, pp. 1227–1237, 2021.

[121] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 22118–22133, 2020.

[122] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," in *International Conference on Learning Representations*, 2018.

[123] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems*, vol. 25, 2012.

[124] D. Zha, K.-H. Lai, S. Huang, Y. Cao, K. Reddy, J. Vargas, A. Nguyen, R. Wei, J. Guo, and X. Hu, "Rlcard: a platform for reinforcement learning in card games," in *International Joint Conference on Artificial Intelligence*, 2021.

[125] D. Zha, J. Xie, W. Ma, S. Zhang, X. Lian, X. Hu, and J. Liu, "Douzero: Mastering doudizhu with self-play deep reinforcement learning," in *International Conference on Machine Learning*, 2021.

[126] D. Zha, K.-H. Lai, K. Zhou, and X. Hu, "Experience replay optimization," in *International Joint Conference on Artificial Intelligence*, 2019.

[127] D. Zha, W. Ma, L. Yuan, X. Hu, and J. Liu, "Rank the episodes: A simple approach for exploration in procedurally-generated environments," in *International Conference on Learning Representations*, 2020.

[128] D. Zha, K.-H. Lai, M. Wan, and X. Hu, "Meta-aad: Active anomaly detection with deep reinforcement learning," in *IEEE International Conference on Data Mining*, 2020.

[129] A. Shaban, C.-A. Cheng, N. Hatch, and B. Boots, "Truncated back-propagation for bilevel optimization," in *International Conference on Artificial Intelligence and Statistics*, pp. 1723–1732, PMLR, 2019.

[130] D. Martinez, D. Zha, Q. Tan, and X. Hu, "Towards personalized preprocessing pipeline search," *arXiv preprint arXiv:2302.14329*, 2023.

[131] D. Zha, L. Feng, B. Bhushanam, D. Choudhary, J. Nie, Y. Tian, J. Chae, Y. Ma, A. Kejariwal, and X. Hu, "Autoshard: Automated embedding table sharding for recommender systems," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2022.

[132] Y. Li, Z. Chen, D. Zha, K. Zhou, H. Jin, H. Chen, and X. Hu, "Autood: Neural architecture search for outlier detection," in *IEEE International Conference on Data Engineering*, 2021.

[133] J. Wang, T. Zhang, S. Liu, P.-Y. Chen, J. Xu, M. Fardad, and B. Li, "Towards a unified min-max framework for adversarial exploration and robustness," *arXiv preprint arXiv:1906.03563*, 2019.

[134] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *International Conference on Learning Representations Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[135] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," in *International Conference on Learning Representations*, 2019.

[136] T. Chen, S. Bian, and Y. Sun, "Are powerful graph neural nets necessary? a dissection on graph classification," *arXiv preprint arXiv:1905.04579*, 2019.

[137] F.-Y. Sun, J. Hoffmann, V. Verma, and J. Tang, "Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization," in *International Conference on Learning Representations*, 2020.

[138] H. Gao, Z. Wang, and S. Ji, "Large-scale learnable graph convolutional networks," in *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1416–1424, 2018.

[139] K. M. Altenburger and J. Ugander, "Monophily in social networks introduces similarity among friends-of-friends," *Nature Human Behaviour*, vol. 2, no. 4, pp. 284–290, 2018.

[140] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[141] L. Jing, P. Vincent, Y. LeCun, and Y. Tian, "Understanding dimensional collapse in contrastive self-supervised learning," in *International Conference on Learning Representations*, 2022.

[142] S. Abu-El-Haija, B. Perozzi, A. Kapoor, N. Alipourfard, K. Lerman, H. Harutyunyan, G. Ver Steeg, and A. Galstyan, "Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing," in *International Conference on Machine Learning*, pp. 21–29, PMLR, 2019.

[143] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *International Conference on Learning Representations*, 2022.

[144] Y. Tian, C. Sun, B. Poole, D. Krishnan, C. Schmid, and P. Isola, "What makes for good views for contrastive learning?," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 6827–6839, 2020.

[145] K. Ding, Z. Xu, H. Tong, and H. Liu, "Data augmentation for deep graph learning: A survey," *ACM SIGKDD Explorations Newsletter*, vol. 24, no. 2, pp. 61–77, 2022.

[146] Y. Zhu, Y. Xu, Q. Liu, and S. Wu, "An empirical study of graph contrastive learning," in *Neural Information Processing Systems Datasets and Benchmarks Track*, 2021.

[147] P. Bielak, T. Kajdanowicz, and N. V. Chawla, "Graph barlow twins: A self-supervised representation learning framework for graphs," *Knowledge-Based Systems*, p. 109631, 2022.

[148] T. Wang and P. Isola, "Understanding contrastive representation learning through alignment and uniformity on the hypersphere," in *International Conference on Machine Learning*, pp. 9929–9939, PMLR, 2020.

[149] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI Magazine*, vol. 29, no. 3, pp. 93–93, 2008.

[150] J. McAuley, C. Targett, Q. Shi, and A. Van Den Hengel, "Image-based recommendations on styles and substitutes," in *ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 43–52, 2015.

[151] A. Sinha, Z. Shen, Y. Song, H. Ma, D. Eide, B.-J. Hsu, and K. Wang, "An overview of microsoft academic service (mas) and applications," in *International Conference on World Wide Web*, pp. 243–246, 2015.

[152] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann, "Tu-dataset: A collection of benchmark datasets for learning with graphs," *arXiv preprint arXiv:2007.08663*, 2020.

[153] X. Gong, C. Yang, and C. Shi, "Ma-gcl: Model augmentation tricks for graph contrastive learning," in *AAAI Conference on Artificial Intelligence*, 2023.

[154] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt, "Efficient graphlet kernels for large graph comparison," in *Artificial intelligence and statistics*, pp. 488–495, PMLR, 2009.

[155] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels.," *Journal of Machine Learning Research*, vol. 12, no. 9, 2011.

[156] P. Yanardag and S. Vishwanathan, "Deep graph kernels," in *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1365–1374, 2015.

[157] B. Adhikari, Y. Zhang, N. Ramakrishnan, and B. A. Prakash, "Sub2vec: Feature learning for subgraphs," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 170–182, Springer, 2018.

[158] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, "graph2vec: Learning distributed representations of graphs," *arXiv preprint arXiv:1707.05005*, 2017.

[159] K. Kwon, H. Na, H. Lee, and N. S. Kim, "Adaptive knowledge distillation based on entropy," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 7409–7413, IEEE, 2020.

[160] J. Zhu, J. Yao, B. Han, J. Zhang, T. Liu, G. Niu, J. Zhou, J. Xu, and H. Yang, "Reliable adversarial distillation with unreliable teachers," in *International Conference on Learning Representations*, 2022.

[161] H. Zhang, D. Chen, and C. Wang, "Confidence-aware multi-teacher knowledge distillation," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 4498–4502, IEEE, 2022.

[162] Y. Gao, H. Yang, P. Zhang, C. Zhou, and Y. Hu, "Graphnas: Graph neural architecture search with reinforcement learning," *arXiv preprint arXiv:1904.09981*, 2019.

[163] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[164] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[165] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," *arXiv preprint arXiv:1811.05868*, 2018.

[166] Q. Zhu, B. Du, and P. Yan, "Self-supervised training of graph convolutional networks," *arXiv preprint arXiv:2006.02380*, 2020.

[167] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, "Masked autoencoders are scalable vision learners," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 15979–15988, IEEE, 2022.

[168] Q. Tan, N. Liu, X. Zhao, H. Yang, J. Zhou, and X. Hu, "Learning to hash with graph neural networks for recommender systems," in *International Conference on World Wide Web*, pp. 1988–1998, 2020.

[169] W. Jin, Y. Ma, X. Liu, X. Tang, S. Wang, and J. Tang, "Graph structure learning for robust graph neural networks," in *ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 66–74, 2020.

[170] J. Li, R. Wu, W. Sun, L. Chen, S. Tian, L. Zhu, C. Meng, Z. Zheng, and W. Wang, "Maskgae: Masked graph modeling meets graph autoencoders," *arXiv preprint arXiv:2205.10053*, 2022.

[171] W. Hu, M. Fey, H. Ren, M. Nakata, Y. Dong, and J. Leskovec, "Ogb-lsc: A large-scale challenge for machine learning on graphs," *arXiv preprint arXiv:2103.09430*, 2021.

[172] J. Li, H. Dani, X. Hu, J. Tang, Y. Chang, and H. Liu, "Attributed network embedding for learning in a dynamic environment," in *ACM International Conference on Information and Knowledge Management*, pp. 387–396, 2017.

[173] Q. Tan, D. Zha, S.-H. Choi, L. Li, R. Chen, and X. Hu, "Double wins: Boosting accuracy and efficiency of graph neural networks by reliable knowledge distillation," 2022.

[174] Q. Tan, N. Liu, X. Huang, S.-H. Choi, L. Li, R. Chen, and X. Hu, "S2gae: Self-supervised graph autoencoders are generalizable learners with graph masking," in *ACM International Conference on Web Search and Data Mining*, pp. 787–795, 2023.

[175] Q. Tan, J. Zhang, J. Yao, N. Liu, J. Zhou, H. Yang, and X. Hu, "Sparse-interest network for sequential recommendation," in *ACM International Conference on Web Search and Data Mining*, pp. 598–606, 2021.

[176] S. Ding, Q. Tan, C.-y. Chang, N. Zou, K. Zhang, N. R. Hoot, X. Jiang, and X. Hu, "Multi-task learning for post-transplant cause of death analysis: A case study on liver transplant," *arXiv preprint arXiv:2304.00012*, 2023.

[177] C.-Y. Chang, J. Yuan, S. Ding, Q. Tan, K. Zhang, X. Jiang, X. Hu, and N. Zou, "Towards fair patient-trial matching via patient-criterion level fairness constraint," *arXiv preprint arXiv:2303.13790*, 2023.