OUT-OF-DISTRIBUTION GENERALIZATION IN GRAPH NEURAL NETWORKS: PROBLEMS, METHODS AND APPLICATIONS

By

Yiqi Wang

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computer Science—Doctor of Philosophy

2022

ABSTRACT

Graphs are one of the most natural representations of many real-world data, such as social networks, chemical molecules, and transportation networks. Graph neural networks (GNNs) are deep neural networks (DNNs) that are specially designed for graphs and have aroused great research interest. Recently, GNNs have been theoretically and empirically proven to be effective in learning graph representations and have been widely applied in many scenarios, such as recommendation and drug discovery. Despite its great success in numerous graph-related tasks, GNNs still face a tremendous challenge in terms of out-of-distribution generalization. Specifically, it has been observed that significant performance gaps for GNNs exist between the training graph set and the test graph set in some graph-related tasks. In addition, graph samples can be very diverse, even though coming from the same dataset. They can be different from each other in not only node attributes but graph structures, which makes the out-of-distribution generalization problem in GNNs more challenging and complex than that in traditional deep learning-based methods. Apart from the out-of-distribution generalization problem, GNNs also come across other kinds of challenges when applied in different application scenarios, such as data sparsity and knowledge transfer in the recommendation task.

In this dissertation, we aim at alleviating the out-of-distribution generalization problem in GNNs. In particular, two novel frameworks are proposed to improve GNN's out-of-distribution generalization ability from two perspectives, i.e., a novel training perspective, and an advanced learning perspective. Meanwhile, we design a novel GNN-based method to solve the data sparsity challenge in the recommendation application. In addition, we propose an adaptive pre-training framework based on the new GNN-based recommendation method and thus increase the abilities of GNNs in terms of generalization and knowledge transfer in the real-world application of recommendations.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my most sincere gratitude and appreciation to my dearest advisor, Dr. Jiliang Tang, for his careful guidance, great patience, constant encouragement, and strong support. During my Ph.D. study, I have come across some unexpected but severe challenges, which made me almost lose faith in the pursuit of the Ph.D. degree. It is Dr.Tang, who lifts me out of the pit of despair, out of the mud and the mire. I have learned so much from him, including the passion, persistence, and skills of research and the faith and courage to make the impossible possible. For me, Dr. Tang is not only an excellent advisor but a dearest friend and a lifelong role model.

I would like to express my deepest appreciation to my committee members, Dr. Anil K. Jain, Dr. Pan-Ning Tan, Dr. Kenneth A. Frank, and Dr. Yunhao Liu for their insightful comments and helpful suggestions. I was very fortunate to collaborate with Dr. Anil K. Jain on several research projects, and I am really impressed by his broad knowledge, brilliant insights, and dedication to research. I took the course Computational Foundations in AI and Machine Learning from Dr. Pan-Ning Tan in the fall of 2019 and it prepares me with useful mathematical knowledge that benefits my Ph.D. study a lot. Dr. Kenneth A. Frank kindly offers numerous insightful suggestions and comments for my research and helps broaden my research vision from a novel perspective. Dr. Yunhao Liu also provides me with lots of valuable advice in terms of research and career. Thank you all again, my dear committee members.

I was fortunate to work with amazing colleagues and mentors: Dr. Chaozhuo Li, Dr. Xing Xie, Dr. Yiding Liu, Dr. Shuaiqiang Wang, and Dr. Dawei Yin. I really enjoyed the wonderful and productive time with you. Thank you all for your careful guidance and valuable suggestions.

Also, I am very grateful to have had so many supportive and encouraging friends and colleagues during my Ph.D. life. I would like to express my thanks to all my dear colleagues and friends from the Data Science and Engineering Lab: Dr. Yao Ma, Wei Jin, Dr. Xiaorui Liu, Dr. Tyler Derr, Dr. Wenqi Fan, Dr. Xiangyu Zhao, Dr. Hamid Karimi, Dr. Zhiwei Wang, Dr. Haochen Liu, Juanhui Li, Yaxin Li, Hua Liu, Jamell Dacon, Wentao Wang, Dr. Xiaoyang Wang, Han Xu, Hongzhi Wen, Harry Shomer, Jie Ren, Jiayuan Ding, Haoyu Han, Yuxuan Wan, Pengfei He, Yingqian Cui, Haitao

Mao, Hang Li, Kaiqi Yang, Dr. Xin Wang, and Dr. Jiangtao Huang. Thank you all for your joyful companionship and support.

Finally, I would like to thank my family, for their unconditional love and support.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Graphs are a kind of common data format, which can naturally denote many real-world data, including biological molecules [9, 39, 126], social networks [19, 122] and transportation networks [111]. There are numerous real-world applications that can be formalized as tasks over graphs. For example, recommendations [108, 25] can be viewed as the link prediction task over user-item bipartite graphs. The property prediction of proteins [9] can be regarded as graph classification tasks over protein graphs. To address these graph tasks, it is important to learn good representations for nodes and graphs. Deep Neural Networks (DNNs) have shown great capacity in representation learning, and have achieved tremendous success in many applications, such as natural language processing [15] and computer vision [33]. Graph Neural Networks (GNNs) are a successful generalization of DNNs over graphs, which have been empirically and theoretically proven to be powerful in graph representation learning. GNNs usually consist of information aggregation among nodes and feature transformation to refine node representations, and graph pooling to summarize a graph representation. In graph tasks, GNNs are typically trained from training samples, and then applied to make predictions among test samples, following the common paradigm adopted in DNNs. One important condition where this paradigm works effectively is that the training samples and the test samples come from the same distribution. However, this condition usually can not be satisfied in real scenarios. Graph data can be especially diverse since samples in graph data can be various in not only the node attributes but also the graph structures. For example, graph samples from one dataset can be very different in terms of graph size, and node samples from one graph can differ from each other greatly in regard to degrees [103]. Such gaps between training samples and test samples can bring significant performance degradation of GNN models. In this dissertation, we mainly focus on solving this challenge in GNNs for graph tasks. Specifically, we tackle this problem from two perspectives, i.e., a training perspective and a learning perspective. In addition, we also focus on addressing the challenges in applying GNNs in real-world recommendation scenarios. In particular, we propose a novel localized GNN-based recommendation method to overcome the data sparsity issue. Furthermore, we propose an adaptive pre-training framework based on the new GNN-based recommendation method and thus increase the abilities of GNNs in terms of generalization and knowledge transfer in the real-world application of recommendation.

1.1 Dissertation Contributions

The major contributions of this dissertation are summarized as follows.

- Numerous recent works have observed that there exist significant performance gaps of GNNs between the training phase and the test phase. We seek to alleviate this problem by designing a novel training paradigm for GNNs. Specifically, instead of training GNNs from the training graph samples and then utilizing the GNNs to directly make predictions for test samples, we propose to further carefully train the GNNs in the test time. We introduce the first test-time training framework for GNNs (GT3) to enhance the model out-of-distribution generalization capacity for the graph classification task. In particular, we design a novel test-time training strategy with a two-level self-supervised learning task to adjust the GNN model for each test graph sample. Also, an effective adaptation constraint is proposed to prevent undesirable excessive feature distortion. Extensive experiments are conducted to validate the effectiveness of the proposed method, and we also provide a theoretical analysis to demonstrate the benefits of test-time training for GNNs.

- The out-of-distribution generalization challenge in GNNs has attracted emerging research interest. We seek to explore and analyze this challenge from the perspective of graph data diversity. We investigate the data diversity in terms of graph structure and their influence on GNNs performance. The investigation inspires us to develop a novel customized graph neural network framework, i.e., Customized-GNN, which can generate a sample-specific model for this graph based on its structure, rather than a unified GNN model for graphs with diverse structures. In addition, the proposed Customized-GNN is very general and flexible that can be applied to numerous existing GNN models. Comprehensive experiments on numerous graph datasets from various domains have verified the effectiveness of the proposed framework.

- GNNs have been widely applied in recommendation tasks and have achieved tremendous success. However, it has been observed that most GNN-based recommendation methods suffer greatly from the data sparsity issue, and unfortunately, this issue is quite common in real-world recommendation scenarios. To solve this limitation, we study a novel perspective to build a GNN-based recommendation method, Localized Graph Collaborative Filtering (LGCF). Specifically, instead of learning embeddings for each user and item, we aim at learning some recommendation-related patterns in the localized graph induced by the target user and item. Extensive experiments on various datasets validate the effectiveness of LGCF, especially in sparse scenarios. Furthermore, empirical results demonstrate that LGCF provides complementary information to the embedding-based recommendation model. Thus, it can be utilized to further boost recommendation performance in other scenarios.

- Data sparsity is one of the most common problems in many real-world applications, such as machine translation, image classification, and recommendation. Pre-training techniques have been proven effective to address data sparsity issues in the domains of natural language processing and computer vision by knowledge transfer. However, it is not trivial to design an effective pre-training framework for the GNN-based method in recommendation tasks. To solve the challenges that different recommendation scenarios not sharing the same users and items and that use-item interaction graphs possessing diverse properties, we design an adaptive GNN pre-training framework (ADAPT) based on LGCF for recommendations, which is capable of capturing the common knowledge among different user-item bipartite graphs and preserving the uniqueness of each graph. The proposed framework does not require transferring user/item embeddings and is able to capture both the common knowledge across different graphs and the uniqueness of each graph simultaneously. Extensive experimental results have demonstrated the effectiveness and superiority of ADAPT.

1.2 Dissertation Structure

The remainder of this dissertation is organized as follows. In Chapter 2, we aim at enhancing the model out-of-distribution generalization capacity for the graph classification task based on a novel training paradigm by introducing the test-time training framework for GNNs. In Chapter 3, we focus on bridging the graph classification gap between the training graphs and test graphs from a new learning perspective by developing a customized graph neural network framework, which can generate a sample-specific model for a test graph based on its structure. In these two chapters, we aim at alleviating the out-of-distribution generalization problem in GNNs from two different perspectives, i.e., the training perspective and the learning perspective. Though from various perspectives, these proposed frameworks are very flexible and can be applied in many existing GNN methods. In Chapter 4, we propose a novel GNN-based recommendation method to solve the data sparsity issue, which is designed to learn useful recommendation information from a localized graph, rather than to learn effective embeddings for all the users and items. In Chapter 5, we further propose an adaptive pre-training framework based on the proposed GNN-based recommendation method, which enables effective knowledge transfer among different recommendation scenarios. In these two chapters, we aim at addressing the practical challenges of applying GNNs in real-world recommendation tasks. In particular, we focus on improving the performance of the GNN-based recommendation methods in data sparsity scenarios and further increase the abilities of GNNs in terms of generalization and knowledge transfer in the real-world application of recommendation. Finally, we conclude the dissertation and discuss further promising research directions in Chapter 6.

CHAPTER 2

TEST-TIME TRAINING FOR GRAPH NEURAL NETWORKS

Graph Neural Networks (GNNs) have made tremendous progress in the graph classification task. However, a performance gap between the training set and the test set has often been noticed. To bridge the gap, in this chapter we introduce the first test-time training framework for GNNs to enhance the model generalization capacity for the graph classification task. In particular, we design a novel test-time training strategy with self-supervised learning to adjust the GNN model for each test graph sample. Experiments on the benchmark datasets have demonstrated the effectiveness of the proposed framework, especially when there are distribution shifts between the training set and the test set. We have also conducted exploratory studies and theoretical analysis to gain a deeper understanding of the rationality of the design of the proposed graph test time training framework (GT3).

2.1 Introduction

Many real-world data can be naturally represented as graphs, such as social networks [19, 122] and molecules [9, 39, 126]. Graph neural networks (GNNs) [7, 62, 68], a successful generalization of deep neural networks (DNNs) over the graph domain, typically refine node representations via information aggregation among nodes and feature transformation that can be summarized as the graph representation via graph pooling operation. GNNs have been empirically and theoretically proven to be effective in graph representation learning, and have achieved revolutionary progress in various graph-related tasks, including node classification [52, 62, 106], graph classification [120, 69, 126] and link prediction [51]. Graph classification is one of the most important tasks on graphs, which aims to predict some properties of graphs. There are lots of real-world scenarios for graph classification, such as predicting molecular property to help drug discovery [17], forecasting the movie type based on the actor/actress connection graphs [35] and predicting the collaboration topic and research interests of an academic collaboration graph [52].

In the graph classification task, we typically train a graph neural network from the training graph

Table 2.1: Graph classification results for *ogbg-molhiv* (Results from the paper [39]).

Method	**ROC-AUC**(%) Training	Validation	Test
GCN	88.65±1.01	83.73±0.78	74.18±1.22
GIN	93.89±2.96	84.1±1.05	75.2±1.30

samples and then utilize the trained model to make predictions for the test graph samples. This training strategy works effectively when the training samples and the test samples come from the same distribution. However, this assumption may not hold in many real scenarios. Often, there could exist gaps between the distributions of the training data and the test data, and the model performance will degrade significantly due to such gaps [101, 75]. For example, as reported by the OBG paper [39] (as shown in Table 2.1), there exist significant gaps between the performance of both the GCN [52] model and the GIN [120] model in the training set and the test set in predicting whether HIV virus replication is inhibited by a molecular. Besides, in the graph domain, graph samples can be very diverse in terms of their graph structures. As demonstrated in Figure 2.1, two graph samples from the same dataset can be very different from each other in terms of structural properties. This also brings in new challenges for the distribution shift problem in graphs. Although recognized by existing works [39, 18], this problem has rarely been explored for GNNs.

Recently, test-time training [63, 101], a newly-proposed paradigm, has been successfully proposed to address the distribution shift issue for images [6, 101]. Specifically, it adapts the model trained on the training set via solving a self-supervised learning task, such as image rotation,

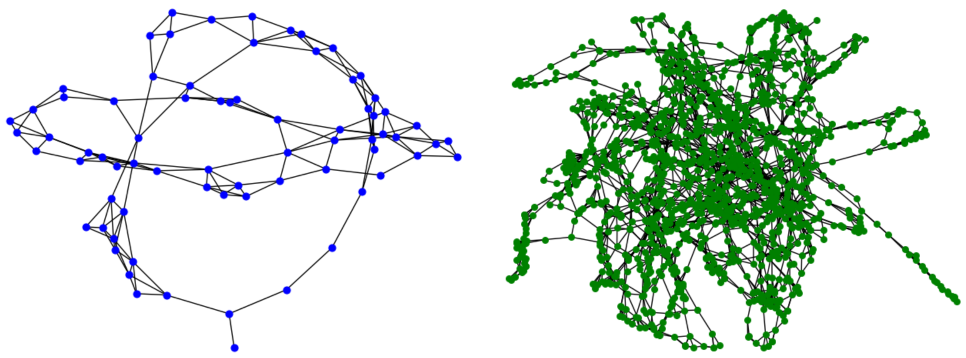


Figure 2.1: Two graph samples from D&D dataset [74], which present very different structure properties.

specifically for every single image at the test time. This paradigm has great potential to solve the distribution graph problem for GNNs since it not only aims at improving the model generalization but also can be designed to specifically adapt a GNN model for each single graph sample with unique structure information.

However, it is challenging to design a test-time training framework for graph neural networks over graph data. First, what self-supervised learning task to design for test-time training in graph data? Graph data usually consists of not only node attribute information but also abundant structure information. Thus, it is important to blend these two kinds of graph information into the design of the self-supervised learning task. Second, how to mitigate the potential severe feature space distortion during the test time? The representation space can be severely tortured in some directions that are undesirable for the main task in the test-time training [63] since the model is adapted solely based on one sample and a self-supervised learning task. This problem can be much more severer for graph data, as graph samples can vary greatly from each other in both the node attributes and the graph structures. Third, as the first work to design a test-time training framework for GNNs, it is desired to understand the rationality of test-time training over GNNs.

In order to solve these challenges, we propose Graph Test-Time Training with Constraint (GT3), a test-time training framework with model adaptation constraints and carefully designed self-supervised learning tasks, especially for graph neural networks. Our key contributions are summarized as follows:

- To the best of our knowledge, this work is the first to design a test-time training framework for the graph neural networks for graph classification tasks.

- We propose to use a two-level self-supervised learning task in the test-time training framework for GNNs, consisting of both the local (node-node level) contrast learning and global (node-graph level) contrast learning, which can fully exploit the attribute and structure information of each graph.

- We have designed an effective adaptation constraint in the test-time training process for

GNNs, which can help prevent excessive feature space distortion and increase the framework's robustness and performance.

- We have conducted exploratory studies and theoretical analysis to understand the rationality of the design of the proposed framework.

2.2 Preliminary

In this section, we will introduce the key notations used in this paper by defining the graph classification problem and reviewing the general structure of Graph Neural Networks (GNNs).

2.2.1 Graph Classification

Graph classification is one of the most common graph-level tasks. It aims at predicting some properties of graphs. There is a wide range of applications for graph classification, such as predicting the molecule properties to assist drug discovery [17] and forecasting whether a protein function as enzymes [126]. Suppose that there is a training set $\mathcal{G} = \{G_1, G_2, \ldots, G_n\}$, consisting of n graphs. For each graph, $G_i \in \mathcal{G}$, its structure information, node attribute information and label information can be denoted as $G_i = (\mathbf{A}_i, \mathbf{X}_i, y_i)$, where $\mathbf{A}_i$ is the graph adjacency matrix, $\mathbf{X}_i$ represents the node attributes and y_i is the label. The goal of graph classification is to learn a classification model $f = (\cdot; \boldsymbol{\theta})$ based on $\mathcal{G}$ which can be used to predict the label of a new graph G'.

2.2.2 Graph Neural Networks

Graph neural networks (GNNs) have successfully extended deep neural networks to graph domains and have been widely applied to numerous graph tasks, including node classification [52, 106], graph classification [126, 120] and link prediction [51]. Generally, graph neural networks consist of several layers, and typically each GNN layer refines node representation by feature transforming, propagating, and aggregating node representation across the graph. According to [24], the node representation update process of node v in the l-th GNN layer of an L-layer GNN can be formulated as follows:

$$\mathbf{a}_v^l = AGGREGATE^l(\mathbf{h}_u^{(l-1)}, u \in \mathcal{N}_v), l \in [L] \tag{2.1}$$

$$\mathbf{h}_v^l = COMBINE^l(\mathbf{h}_v^{(l-1)}, \mathbf{a}_v^l), l \in [L], \tag{2.2}$$

where $AGGREGATE$ is the aggregation function aggregating information from neighborhood nodes $u \in \mathcal{N}_v$, and $COMBINE$ is the combination function updating the node representation in the l-th layer based on the aggregated information a_v^l and the node representation of v from the previous layer $(l-1)$. For simplicity, we summarize the node representation update process in the l-th GNN layer, consisting of Eqs 2.1 and 2.2, as follows:

$$\mathbf{H}^l = GNN(\mathbf{A}, \mathbf{H}^{(l-1)}; \boldsymbol{\theta}_l), l \in [L], \tag{2.3}$$

where $\mathbf{A}$ is the adjacency matrix, $\mathbf{H}^l$ denotes node representations of all the nodes in the l-th GNN layer, and $\boldsymbol{\theta}_l$ indicates the learnable parameters in the l-th layer. Note that $\mathbf{H}^0$ is the input node attributes $\mathbf{X}$.

For graph classification, an overall graph representation is summarized from the L-th GNN layer based on the node representations by a readout function, which will be utilized by the prediction layer to predict the label of the graph:

$$\mathbf{h}_G = READOUT(\mathbf{H}^L), \tag{2.4}$$

2.3 The Proposed Test-time Training Framework for GNNs

In this section, we introduce the proposed framework of Graph Test-Time Training with Constraint (GT3) and detail its key components. We first discuss the overall process, then detail the two-level SSL task and finally describe the adaptation constraint to mitigate the potential issue of feature distortion in GT3.

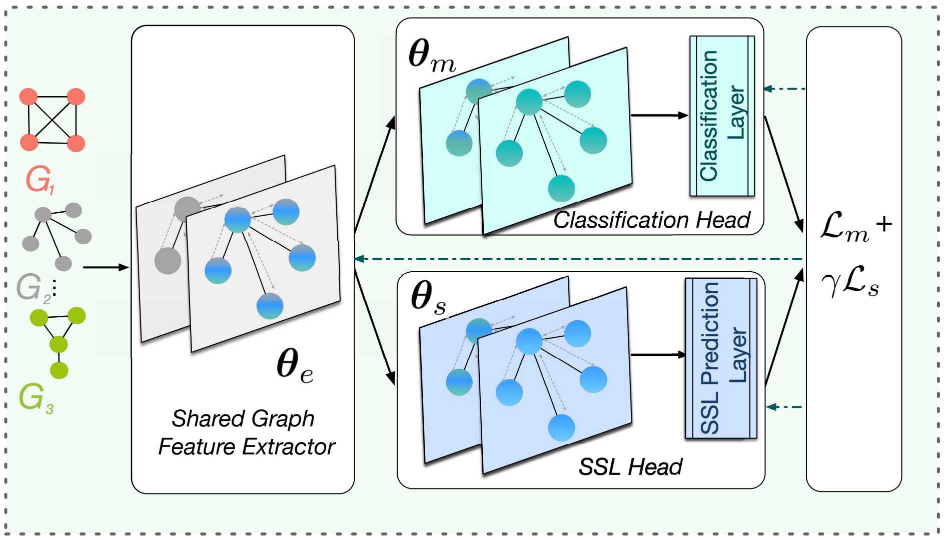
2.3.1 An Overview

Suppose that the graph neural network for the graph classification task consists of L GNN layers, and the learnable parameters in the l-th layer are denoted as $\boldsymbol{\theta}_l$. The overall model parameters can be denoted as $\boldsymbol{\theta}_{main} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_L, \boldsymbol{\delta}_m)$, where $\boldsymbol{\delta}_m$ is the parameters in the prediction layer. In this work, we mainly focus on the graph classification task, which is named as *the main task* in the test-time training framework. Given the training graph set $\mathcal{G} = (G_1, G_2, \ldots, G_n)$ and its corresponding graph label set $\mathcal{Y} = (y_1, y_2, \ldots, y_n)$. The main task is to solve the following empirical risk minimization problem:

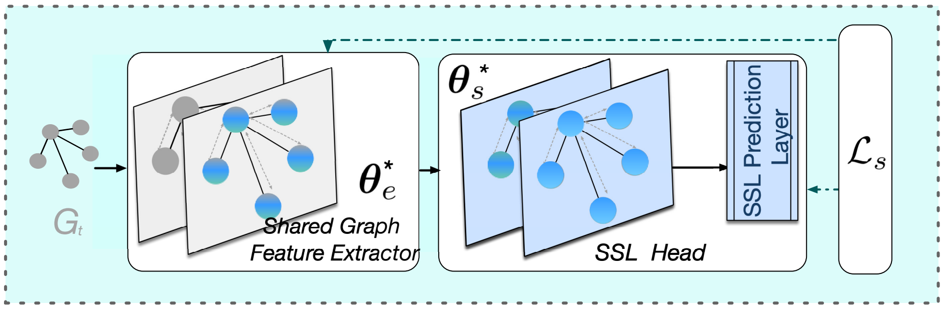
$$\min_{\boldsymbol{\theta}_{main}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_m(\mathbf{A}_i, \mathbf{X}_i, y_i; \boldsymbol{\theta}_{main}), G_i \in \mathcal{G}, \tag{2.5}$$

where $\mathcal{L}_m(\cdot)$ denotes the loss function for the main task.

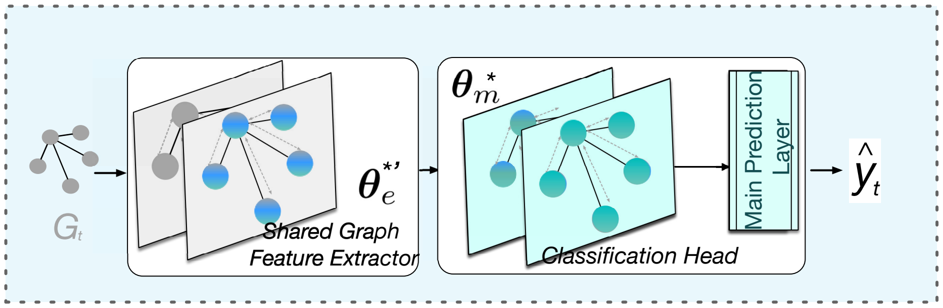
In addition to the main task, self-supervised learning (SSL) tasks also play a vital role in the test-time training framework. As informed by [63], the SSL tasks should be as much information as possible. Also, the more correlated the SSL task and the main task are, the more likely the test-time training will benefit the main task. Given the great success of contrastive learning, methods have been achieved as auxiliary tasks in graph-related tasks, in this work, we carefully design a two-level graph contrastive learning task as the SSL task (details in Section 2.3.2). For simplicity, the loss for any self-supervised learning task is denoted as $\mathcal{L}_s(\cdot)$. With the main task and the SSL task, an overall of the proposed framework GT3 is shown in Figure 2.2. It consists of three key processes: (1) the training process (Figure 2.2(a)); (2) the test-time training process (Figure 2.2(b)) and (3) the inference process (Figure 2.2(c)).



(a) The Training Process.



(b) Test-time Training Process.



(c) Inference Process.

Figure 2.2: An Overview of the Proposed Framework GT3.

The Training Process. Given the training graphs $\mathcal{G}$, the training process is to train the model parameters with the main task and the SSL task. The key challenge is how to integrate the main task and the SSL task. To tackle this challenge, we conduct an empirical study where we train two GNN models separately by the main task and the SSL task, and we found that these two GNN models will extract similar features by their first few layers. More details can be found at Section 2.5.2. This empirical finding paves us a way to integrate the main task and the SSL task: we allow them to share the parameters in the first few layers as shown in Figure 2.2(a). We denote the shared model parameters as $\boldsymbol{\theta}_e = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_K)$, where $K \in \{1, 2, \ldots, L\}$. The shared component is named *shared graph feature extractor* in this work. The model parameters that are specially for the main task are denoted as $\boldsymbol{\theta}_m = (\boldsymbol{\theta}_{K+1}, \ldots, \boldsymbol{\theta}_L, \delta_m)$, described as *the graph classification head*. Correspondingly, the SSL task has its own *self-supervised learning head*, denoted as $\boldsymbol{\theta}_s = (\boldsymbol{\theta}'_{K+1}, \ldots, \boldsymbol{\theta}'_L, \boldsymbol{\delta}_s)$, where $\boldsymbol{\delta}_s$ is the prediction layer for the SSL task. To summarize, the model parameters for the main task can be denoted as $\boldsymbol{\theta}_{main} = (\boldsymbol{\theta}_e, \boldsymbol{\theta}_m)$, the model parameters for the SSL task can be represented as $\boldsymbol{\theta}_{self} = (\boldsymbol{\theta}_e, \boldsymbol{\theta}_s)$, and these for the whole framework can be denoted as $\boldsymbol{\theta}_{overall} = (\boldsymbol{\theta}_e, \boldsymbol{\theta}_m, \boldsymbol{\theta}_s)$. In the training process, we train the whole framework by minimizing a weighted combination loss of $\mathcal{L}_m(\cdot)$ and $\mathcal{L}_s(\cdot)$ as follows:

$$\begin{aligned}\min_{\boldsymbol{\theta}_e, \boldsymbol{\theta}_m, \boldsymbol{\theta}_s} \frac{1}{n} \sum_{i=1}^{N} &\mathcal{L}_m(\mathbf{A}_i, \mathbf{X}_i, y_i; \boldsymbol{\theta}_e, \boldsymbol{\theta}_m) + \\ &\gamma \mathcal{L}_s(\mathbf{A}_i, \mathbf{X}_i, y_i; \boldsymbol{\theta}_e, \boldsymbol{\theta}_s), G_i \in \mathcal{G}, \end{aligned} \tag{2.6}$$

where γ is a hyper-parameter balancing the main task and the self-supervised task in the training process.

The Test-time Training Process. Given a test graph G_t, the test-time training process will fine-tune the model via the SSL task as illustrated in Figure 2.2(b). Specifically, given a test graph sample $G_t = (\mathbf{A}_t, \mathbf{X}_t)$, the *shared graph feature extractor* and the *self-supervised learning head* are fine-tuned by minimizing the SSL task loss as follows:

$$\min_{\boldsymbol{\theta}_e, \boldsymbol{\theta}_s} \mathcal{L}_s(\mathbf{A}_t, \mathbf{X}_t; \boldsymbol{\theta}_e, \boldsymbol{\theta}_s), \tag{2.7}$$

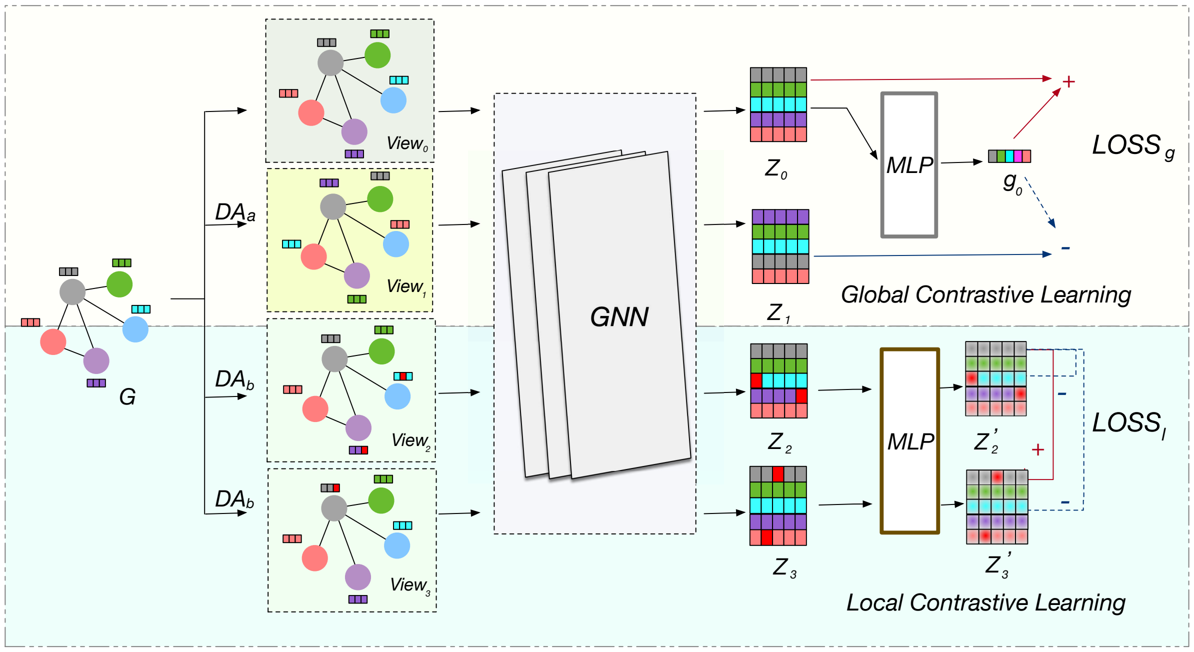


Figure 2.3: The Self-supervised Learning Task for GT3.

Suppose that $\boldsymbol{\theta}^*_{overall} = (\boldsymbol{\theta}^*_e, \boldsymbol{\theta}^*_m, \boldsymbol{\theta}^*_s)$ is the optimal parameters from minimizing the overall loss of Eq 2.6 from the training process. In the test process, given a specific test graph sample G_t, the proposed framework further updates $\boldsymbol{\theta}^*_e$ and $\boldsymbol{\theta}^*_s$ to $\boldsymbol{\theta}^{*'}_e$ and $\boldsymbol{\theta}^{*'}_s$ by minimizing the SSL loss of Eq 2.7 over the test graph sample G_t.

The Inference Process. The inference process is to predict the label of the test graph G_t based on the finetuned model by the test-time training as demonstrated in Figure 2.2(c). In particular, the label of G_t is predicted via the GNN model $f(\cdot; \boldsymbol{\theta}^{*'}_e, \boldsymbol{\theta}^*_m)$ as:

$$\hat{y}_t = f(\mathbf{X}_t, \mathbf{A}_t; \boldsymbol{\theta}^{*'}_e, \boldsymbol{\theta}^*_m). \tag{2.8}$$

2.3.2 Self-supervised Learning Task for GT3

An appropriate and informative SSL task is the key to the success of test-time training [63]. However, it is challenging to design an appropriate SSL task for test-time training for GNNs. First, graph data is intrinsically different from images, some common properties such as rotation invariance do not exist in graph data, thus most SSL tasks commonly adopted by existing test-time training are invalid in graph data. Second, most existing SSL tasks for graph classification are intra-graph contrastive learning based on the difference among diverse graph samples [99, 31]. These are not applicable for the test-time training where we aim at specifically adapting the model for every single graph during the test time. Third, graph data usually consists of not only node attribute information but abundant and unique structure information. It is important to fully leverage these two kinds of graph information into the design of the SSL task since an informative SSL task is one of the keys to the success of test-time training.

Motivated by the success of contrastive learning in graph domains, we propose to use a two-level SSL task from both local and global perspectives for GT3, which fully exploits the graph information from both the node-node level and node-graph level. The proposed self-supervised learning task is based on graph characteristics, instead of the properties in Euclidean data. In addition, the proposed SSL task is not built up based on the differences among distinct graphs, thus it can be naturally applied to a single graph. We empirically validate that the global (node-graph level) contrastive learning task and the local (node-node level) contrastive learning task in the proposed two-level SSL task is necessary for GT3 since they are complementary to each other across datasets. More details about this empirical study can be found in Section 2.5.2. Next, we will detail the global contrastive learning task and the local contrastive learning task.

2.3.2.1 Global Contrastive Learning

Global contrastive learning aims at helping node representation capture global information of the whole graph. Overall, global contrastive learning is based on maximizing the mutual information between the local node representation and the global graph representation. Specifically, as shown in Figure 2.3, given an input graph sample G, different views can be generated via various types of data augmentations. For global contrastive learning, two views are adopted: one is the raw view $View_0$, where no changes are made on the original graph; the other one is augmented view $View_1$, where node attributes are randomly shuffled among all the nodes in a graph. With these two graph views, we can get two corresponding node representations $\mathbf{Z}_0$ and $\mathbf{Z}_1$ via a shared GNN model.

After that, a global graph representation $\mathbf{g}_0$ can be summarized from the node representation matrix $\mathbf{Z}_0$ from the raw view $View_0$ via a multiplayer perceptron (MLP). In our work, a single-liner layer model is adopted as the MLP.

Following [107], the positive samples in the global contrastive learning consist of node-graph representation pairs, where both the node representation and graph representation come from the raw view $View_0$. The negative samples consist of node-graph representation pairs where node representations come from $View_1$ and graph representation comes from $View_0$. A discriminator $\mathcal{D}$ is employed to compute the probability score for each pair, and the score should be higher for the positive pair and lower for the negative pair. In our work, we set $\mathcal{D}(\mathbf{Z}_{si}, \mathbf{g_0}) = \mathbf{Z}_{si} * \mathbf{g}_0$, where $\mathbf{Z}_{\mathbf{s}i}$ denotes the representation of node i from $View_s$. The objective function for global contrastive learning is summarized as follows:

$$\mathcal{L}_g = -\frac{1}{2N}(\sum_{i=1}^{N}(log\mathcal{D}(\mathbf{Z}_{0i}, \mathbf{g}_0) + log(1 - \mathcal{D}(\mathbf{Z}_{1i}, \mathbf{g}_0))), \tag{2.9}$$

where N denotes the number of nodes in the input graph.

2.3.2.2 Local Contrastive Learning

In global contrastive learning, the model aims at capturing the global information of the whole graph into the node representations. In other words, the model attempts to learn an invariant graph representation from a global level and blend it into the node representations. However, graph data consists of nodes with various attributes and distinct structural roles. To fully exploit the structure information of a graph, in addition to the invariant graph representation from a global level, it is also important to learn invariant node representations from a local level. To achieve this, we propose local contrastive learning, which is based on distinguishing different nodes from different augmented views of a graph.

As shown in Figure 2.3, given an input graph G, we can generate two views of the graph via data augmentation. Since the local contrastive learning is to distinguish whether two nodes from different views are the same node of the input graph, the adopted data augmentation should not make large changes on the input graph. To meet this requirement, we adopt two adaptive graph data mechanisms – adaptive edge dropping and adaptive node attribute masking. Specifically, the edges are dropped based on the edge importance in the graph. The more important an edge is, the less likely it would be dropped. Likewise, the node attributes are masked based on the importance of each dimension of node attributes. The more important the attribute dimension is, the less likely it would be masked out. Intuitively, the higher degree a node holds, the more important role it plays in the whole graph. Therefore, in this work, the importance of both the edges and node attributes are computed based on the node degree. In particular, the importance score of an edge is computed as the average degree of its connected nodes and the importance score of an attribute is the average of the products of its norm over all nodes and the degree of the corresponding node in the whole graph.

After the described data augmentation, two views of an input graph are generated and serve as the input of the shared GNN model. The outputs are two node representation matrices $\mathbf{Z}_2$ and $\mathbf{Z}_3$ corresponding to two views. Note that the basic goal of local contrastive learning is to distinguish whether two nodes from augmented views are the same node in the input graph. Therefore, $(\mathbf{Z}_{2i}, \mathbf{Z}_{3i})$, $(i \in \{1, \dots, N\})$ denotes a positive pair, where N is the number of nodes. $(\mathbf{Z}_{2i}, \mathbf{Z}_{3j})$ and $(\mathbf{Z}_{2i}, \mathbf{Z}_{2j})$ $(i, j \in \{1, \dots, N\}$ and $i \neq j)$ denote an intra-view negative pair and an inter-view negative pair, respectively. Inspired by InfoNCE [79, 141], the objective for a positive node pair $(\mathbf{Z}_{2i}, \mathbf{Z}_{3i})$ is defined as follow:

$$\mathfrak{l}_c\left(\mathbf{Z}_{2i}, \mathbf{Z}_{3i}\right) = log \frac{h(\mathbf{Z}_{2i}, \mathbf{Z}_{3i})}{h(\mathbf{Z}_{2i}, \mathbf{Z}_{3i}) + \sum_{j \neq i} h(\mathbf{Z}_{2i}, \mathbf{Z}_{3j}) + \sum_{j \neq i} h(\mathbf{Z}_{2i}, \mathbf{Z}_{2j})}, \tag{2.10}$$

where $h(\mathbf{Z}_{2i}, \mathbf{Z}_{3j}) = e^{cos\left(g(\mathbf{Z}_{2i}), g(\mathbf{Z}_{3j})\right)/\tau}$, $cos()$ denotes the cosine similarity function, τ is a temperature parameter and $g()$ is a two-layer perceptron (MLP) to further enhance the model expression power. The node representations refined by this MLP are denoted as $\mathbf{Z}_2^{'}$ and $\mathbf{Z}_3^{'}$, respectively.

Apart from the contrastive objective described above, a decorrelation regularizer has also been added to the overall objective function of the local contrastive learning, in order to encourage different representation dimensions to capture distinct information. The decorrelation regularizer is applied over the refined node representations $\mathbf{Z}_2^{'}$ and $\mathbf{Z}_3^{'}$ as follows:

$$\ell_d(\mathbf{Z}_2^{'}) = \left\| \mathbf{Z}_2^{'T} \mathbf{Z}_2^{'} - I \right\|_F^2 . \tag{2.11}$$

The overall objective loss function of local contrastive learning is denoted as follows:

$$\mathcal{L}_l = -\frac{1}{2N} \sum_{i=1}^{N} (\ell_c(\mathbf{Z}_{2i}, \mathbf{Z}_{3i}) + \ell_c(\mathbf{Z}_{3i}, \mathbf{Z}_{2i})) - \frac{\beta}{2} (\ell_d(\mathbf{Z}_2^{'}) + \ell_d(\mathbf{Z}_3^{'})), \tag{2.12}$$

where β is the balance parameter.

2.3.2.3 The Overall Objective Function

The overall objective loss function for the SSL task for GT3 is a weighted combination of the global contrastive learning loss and the local contrastive loss as follows:

$$\mathcal{L}_s = \mathcal{L}_g + \alpha \mathcal{L}_l, \tag{2.13}$$

where α is the parameter balancing the local contrastive learning and the global contrastive learning. By minimizing the SSL loss defined in Eq 2.13, the model not only captures the global graph information but also learns the invariant and key node representations.

2.3.3 Adaptation Constraint

Directly applying test-time training could lead to severe representation distortion, since the model could overfit to the SSL task for a specific test sample [63], which harms the model performance for the main task. This problem may be riskier for test-time training for graph data since graph samples can vary from each other in not only node attributes but also graph structures. To mitigate this issue, we propose to add an adaptation constraint into the objective function during the test-time training process. The core idea is to put a constraint over the embedding spaces outputted by the *shared graph feature extractor* between the training graph samples and the test graph sample. In such a way, there is an additional constraint for the embedding of the test graph sample to be close to the embedding distribution of training graph samples which is generated with the guidance of the main task, in addition to just fitting the SSL task.

The *shared graph feature extractor* consists of the first K GNN layers. Let us denote the node embeddings outputted by the *shared graph feature extractor* for the training graphs $\{G_1, G_2, \ldots, G_n\}$

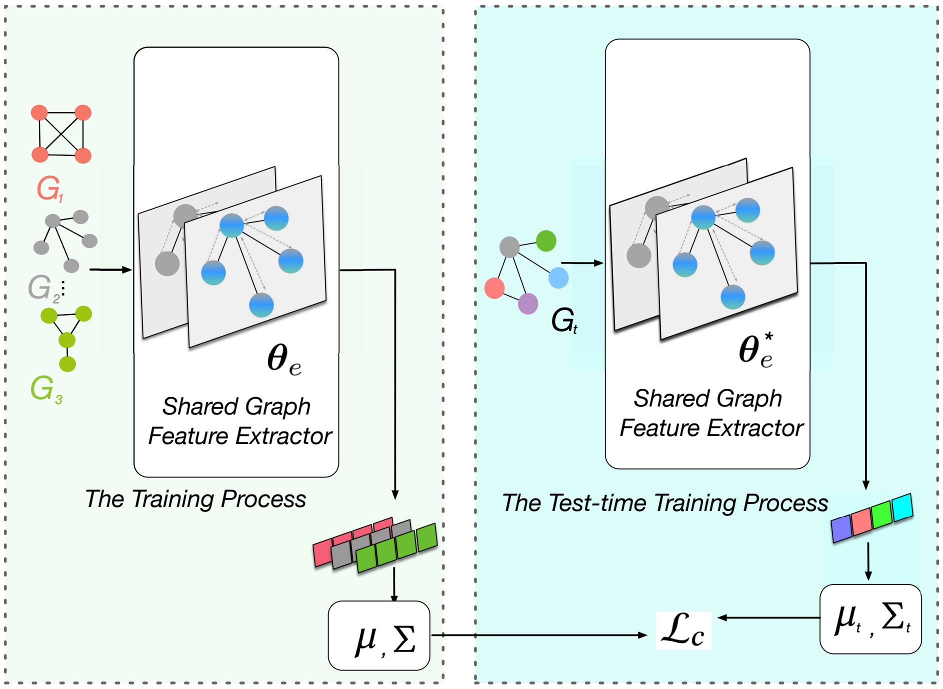


Figure 2.4: The Adaptation Constraint in the Test-time Training Process.

in the training process as $\{\mathbf{H}_1^K, \mathbf{H}_2^K, \ldots, \mathbf{H}_n^K\}$. Once the training process completes, we first get graph embeddings via a read-out function $\mathbf{h}_i^K = READOUT(\mathbf{H}_i^K)$, and then summarize two statistics of these graph embeddings: the empirical mean $\mu = \frac{1}{n}\sum_i^n \mathbf{h}_i^K$ and the covariance matrix $\mathbf{\Sigma} = \frac{1}{n-1}(\mathbf{H}^{K^T}\mathbf{H}^K - (\mathbf{I}^T\mathbf{H}^K)^T(\mathbf{I}^T\mathbf{H}^K))$, where $\mathbf{H}^K = \{\mathbf{h}_1^{K^T}, \ldots, \mathbf{h}_n^{K^T}\}$. During the test-time training process, given an input test graph G_t, we also summarize embedding statistics of G_t and its augmented views and denote them as μ_t and $\mathbf{\Sigma}_t$. The adaptation constraint is to force the embedding statistics of the test graph sample to be close to these of training graph samples, which can be formally defined as:

$$\mathcal{L}_c = \|\mu - \mu_t\|_2^2 + \|\mathbf{\Sigma} - \mathbf{\Sigma}_t\|_F^2 \, . \tag{2.14}$$

2.4 Theoretical Analysis

In this section, we explore the rationality of test-time training for GNNs from a theoretical perspective. The goal is to demonstrate that the test-time training framework can be beneficial for GNNs in the graph classification task. Next, we first describe some basic notions, and then develop two supportive theorems to achieve the goal.

For a graph classification task, suppose that there are C classes and we use a L-layer simplified graph convolution (SGC) [115] with the sum-pooling layer to learn graph representation, i.e.,

$$\mathbf{g} = f_{\text{GNN}}(\mathbf{A}, \mathbf{X}; \boldsymbol{\theta}) = \text{pool}(\mathbf{A}^L \mathbf{X} \mathbf{W}_1) \mathbf{W}_2 = \mathbf{1}^{\mathbf{T}} \mathbf{A}^{\mathbf{L}} \mathbf{X} \boldsymbol{\theta}, \tag{2.15}$$

where $\mathbf{A}$ and $\mathbf{X}$ are the input graph adjacency matrix and node attributes, $\mathbf{1}$ is a vector with all elements as 1, and $\boldsymbol{\theta} = \mathbf{W}_1 \mathbf{W}_2$ denotes the model parameters. Note that we use SGC for this theoretical analysis since SGC has a similar filtering pattern as GCN but its architecture is simpler. Then, the $softmax$ function is applied over the graph representation $\mathbf{g}$ to get the prediction probability $\mathbf{p}_c$:

$$\mathbf{p}_c = \frac{e^{\mathbf{g}_c}}{\sum_{i=1}^{C} e^{\mathbf{g}_i}}, \tag{2.16}$$

where $\sum_{c=1}^{C} \mathbf{p}_c = 1$. The cross-entropy loss is adopted as the optimization objective for the SGC model, i.e.,

$$\mathcal{L}(\mathbf{A}, \mathbf{X}, y; \boldsymbol{\theta}) = -\sum_{c=1}^{C} 1_{y=c} \log(\mathbf{p}_c), \tag{2.17}$$

where y is the graph ground-truth label.

Theorem 1. *Let $\mathcal{L}(\mathbf{A}, \mathbf{X}, y; \boldsymbol{\theta})$ be defined based on Eqs 2.15 and 2.16, 2.17, i.e., the cross-entropy loss for softmax classification for a L-layer SGC with sum-pooling for a graph $\mathbf{A}, \mathbf{X}, y$. It is convex and β-smooth in $\boldsymbol{\theta}$, and $\|\nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{A}, \mathbf{X}, y; \boldsymbol{\theta})\| \leq G'$ for all $\boldsymbol{\theta}$, where G' is a positive constant.*

Proof Sketch. We demonstrate that the cross-entropy loss for softmax classification is convex and β-smooth by proving its Hessian Matrix is positive semi-definite and the eigenvalues of the Hessian Metrix are smaller than a positive constant. As f_{GNN} defined in Eq 2.15 is a linear transformation mapping function, thus it will not change these properties, which completes the proof of Theorem 1. Proof details can be found in Appendix A.1.1 .

Theorem 2. *Let $\mathcal{L}_m(x, y; \boldsymbol{\theta})$ denote a supervised task loss on a instance x, y with parameters $\boldsymbol{\theta}$, and $\mathcal{L}_s(x; \boldsymbol{\theta})$ denotes a self-supervised task loss on a instance x also with parameters $\boldsymbol{\theta}$. Assume that for all x, y, $\mathcal{L}_m(x, y; \boldsymbol{\theta})$ is differentiable, convex and β-smooth in $\boldsymbol{\theta}$, and both*

$\|\nabla_{\boldsymbol{\theta}}\mathcal{L}_m(x, y; \boldsymbol{\theta})\|, \|\nabla_{\boldsymbol{\theta}}\mathcal{L}_s(x; \boldsymbol{\theta})\| \leq G$ for all $\boldsymbol{\theta}$, where G is a positive constant. With a fixed learning rate $\eta = \frac{\epsilon}{\beta G^2}$, for every x, y such that

$$\langle \nabla_{\boldsymbol{\theta}}\mathcal{L}_m(x, y; \boldsymbol{\theta}), \nabla_{\boldsymbol{\theta}}\mathcal{L}_s(x; \boldsymbol{\theta})\rangle > \epsilon, \tag{2.18}$$

we have

$$\mathcal{L}_m(x, y; \boldsymbol{\theta}) > \mathcal{L}_m(x, y; \boldsymbol{\theta}(x)), \tag{2.19}$$

where $\boldsymbol{\theta}(x) = \boldsymbol{\theta} - \eta\nabla_{\boldsymbol{\theta}}\mathcal{L}_s(x; \boldsymbol{\theta})$.

The detailed proof is provided in Appendix A.1.2. *Theorem* 2 suggests that if the objective function of the main task is differentiable, convex and β-smooth, there exists a SSL task meeting some requirements and the test-time training via this SSL task can make the main task loss decrease.

Remark. From *Theorem* 1, we know that the objective loss function for the SGC model for graph classification is differentiable, convex and β-smooth. Combined with *Theorem* 2, we can conclude that test-time training can be beneficial for the GNN performance on graph classification.

2.5 Experiments

In this section, we validate the effectiveness and explore the rationality of the proposed GT3 via empirical studies. In the following subsections, the experimental settings are first introduced. Then, we conduct a layer-wise feature space exploration of GNN models trained for different tasks to further validate the rationality of the design of GT3. Next, the classification performance comparison of the proposed GT3 and baseline methods is discussed. Finally, we perform an ablation study and hyper-parameter sensitivity exploration.

2.5.1 Experimental Settings

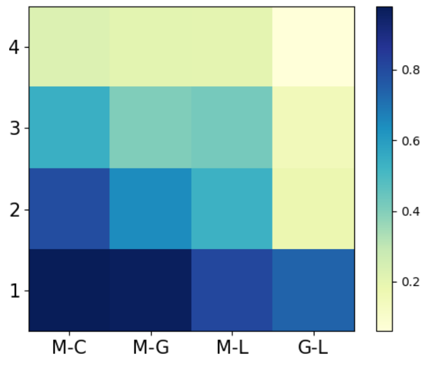
In this study, we implement the proposed GT3 on two well-known GNN models: the Graph Convolutional Network (GCN) [52] and the Graph Isomorphism Network (GIN) [120]. Empirically, we implement four-layer GCN models and three-layer GIN models with max-pooling. The layer normalization is adopted in the implementations. The hyper-parameters are selected based on the validation set from the ranges listed as follows:

- Hidden Embedding Dimension: 32,64,128
- Batch Size: 8,16,32,64
- Learning Rate: [1e-5, 5e-3]
- Dropout Rate: 0.0,0.1,0.2,0.3,0.4,0.5

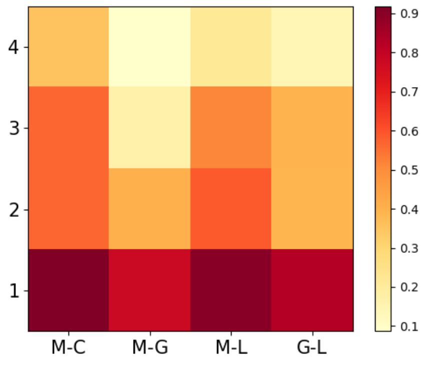
For the balance hyper-parameters α, β, and γ in the SSL task, we determine their values based on the joint training of the SSL task and the main task on the validation set. As for performance evaluation, we conduct experiments on data splits generated by 3 random seeds and model initialization of 2 random seeds on DD, ENZYMES, and PROTEINS, and report the average and variance values of classification accuracy. For ogbg-molhiv, we do an empirical study on the fixed official data split with model initialization of 2 random seeds and report the average performance in terms of ROC-AUC. We conduct experiments on four common graph datasets as follows:

- **DD** [74]: a dataset consists of protein structures of two categories, where each node denotes an amino aid.
- **ENZYMES** [74]: it consists of enzymes of six categories, where each enzyme is represented as its tertiary structure.
- **PROTEINS** [74]: a dataset contains protein structures of two categories, where each node denotes a secondary structure element.
- **ogbg-molhiv** [39]: it includes molecules of two categories (carrying HIV virus or not), where each node is an atom.

In order to simulate the scenario where the test set distribution is different from the training set, graph samples from DD, ENZYMES, and PROTEINS are split into two groups based on their graph size (number of nodes). Next, we randomly select 80% of the graphs from the small group as the training set and the other 20% of the graphs as the validation set. The test set consists of



(a) ENZYMES.



(b) PROTEINS.

Figure 2.5: CKA Values of Representation from Different Layers for Various Tasks. Note that "M" denotes the graph classification task; "C" indicates the proposed GT3 SSL task; "G" is the global contrastive learning task, and "L" is the local contrastive learning task.

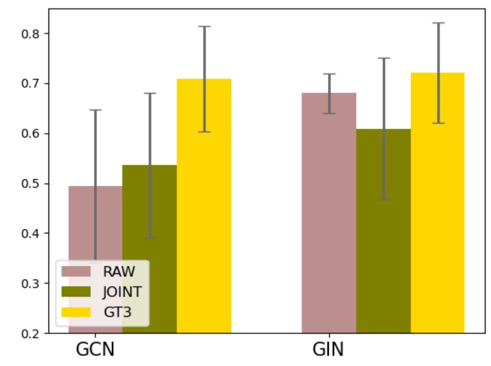
graphs randomly chosen from the larger group and the number of test graphs is the same as that of the validation set. For the ogbg-molhiv, we adopt the official scaffold splitting, which also aims at separating graphs with different structures into different sets. For simplicity, these data splits described above are denoted as *OOD* data split. To validate the reasonability of the *OOD* data split, we examine the training, validation, and test performance of GCN, and the results are shown in Table 2.2. Note that the performance in ogbg-molhiv is copied from [39]. From the table, we indeed observe a big performance gap.

Table 2.2: Graph Classification Performance Gap of GCN on Training Set, Validation Set, and Test Set based on *OOD* Data Split. Note that the performance metric for ogbg-molhiv is **ROC-AUC**(%) and that for others is **Accuracy**(%).

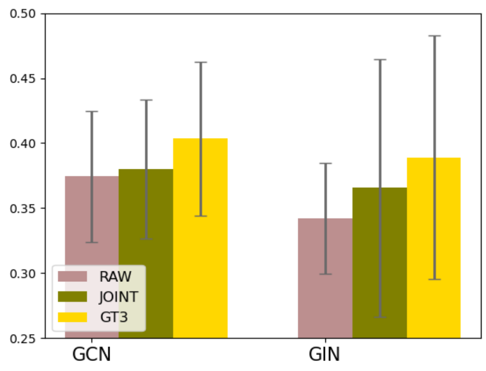
Dataset	**Graph Classification Performance**		
	Training	Validation	Test
DD	92.8±7.13	73.82 ±3.36	49.43 ±15.30
ENZYMES	87.43±12.34	65.22 ±3.76	37.43 ±5.03
PROTEINS	77.5±2.07	77.78 ±1.93	68.46 ±7.17
ogbg-molhiv	88.65±1.01	83.73±0.78	74.18±1.22

2.5.2 Layer-wise Representation Exploration

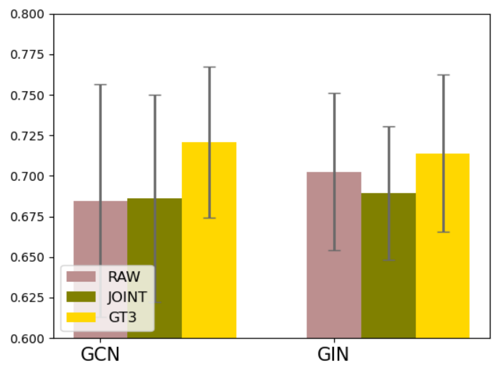
Test-time training framework is designed to share the first few layers of Convolutional Neural Networks (CNNs) in the existent work. This design is very natural since it is well received that the first few layers of CNNs capture similar basic patterns even when they are trained for different tasks. However, it is unclear how GNNs work among different tasks. To answer this question, we explore the representation similarity of GNNs trained for different tasks. To be specific, we train four GCN models with exactly the same architectures on ENZYMES and PROTEIN for four different tasks: the graph classification task defined in Eq 2.5, the proposed GT3 SSL task with loss described in Eq 2.13, the SSL tasks with the local contrastive loss (Eq 2.12) and the global contrastive loss (Eq 2.9). Then we leverage Centered Kernel Alignment (CKA) [54] as a similarity index to evaluate the representation similarity of different GCN models layer by layer.



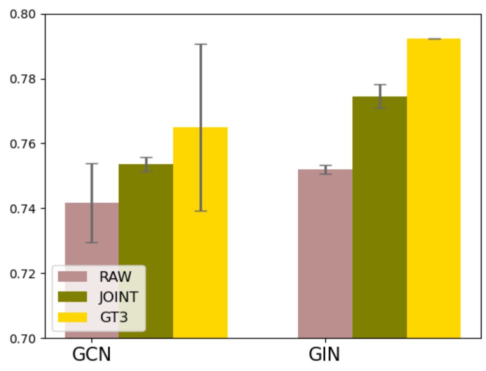
(a) DD.



(b) ENZYMES.



(c) PROTEINS.



(d) ogbg-molhiv.

Figure 2.6: Performance Comparison of GCN and GIN on Four Datasets based on *OOD* Data Split. Note that the performance metric for ogbg-molhiv is **ROC-AUC**(%) and that for others is **Accuracy**(%).

The similarity results are demonstrated in Figure 2.5, where “M" denotes the graph classification task, “C" indicates the proposed GT3 SSL task, “G" is the global contrastive learning task and “L" is the local contrastive learning task. "M-C" represents the representation pair of "M" and "C" to be compared. The representation pair for other tasks are denoted similarly. We can make the following observations. First, the representations for different tasks tend to be more similar to each other in the lower layer. Second, the representation similarity for the same task pair may be different in different datasets. The representations of the graph classification task and the global contrastive learning task are more similar to each other compared to that of the graph classification task and the local contrastive learning task in ENZYMES, while the observation is the opposite in PROTEINS. Third, the representation similarity between local contrastive learning and global contrastive learning is relatively lower, which reveals that they may learn knowledge that is complementary to each other. These observations motivate us (1) to share the first few layers of GNNs in test-time training; and (2) to develop the two-level SSL task for GT3, which is capable of capturing graph information from both the local and global levels.

2.5.3 Performance Comparison

In this subsection, we compare the graph classification performance of GCN and GIN in three mechanisms: (1) *RAW* model is trained only by the main task, (2) *JOINT* model is jointly trained by the main task and the proposed two-level SSL task, and (3) *GT3* model is learned by test-time training. The comparison results are shown in Figure 2.6. Note that the performance of *RAW* in ogbg-molhiv is copied from [39]. From the figure, we can observe that the joint training of the proposed two-level SSL task and the main task can slightly improve the model performance in some cases. However, the proposed *GT3* is able to consistently perform better than both *RAW* and *JOINT*, which demonstrates the effectiveness of the proposed *GT3*. To further demonstrate the performance improvement from *GT3*, we summarize the relative performance improvement of *GT3* over *RAW*. The results are shown in Table 2.3 where the performance improvement can be up to 43.3%. In addition to the performance comparison on the *OOD* data split, we have also conducted a performance comparison on a random data split, where we randomly split the dataset into 80%/10%/10% for the training/validation/test sets, respectively. The results demonstrate that

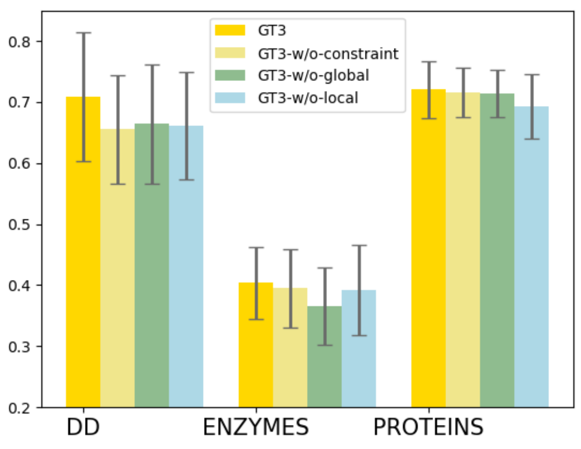
Table 2.3: Relative Performance Improvement of *GT3* over *RAW*.

Models	**Datasets**			
	DD	ENZYMES	PROTEINS	ogbg-molhiv
GCN	+43.3%	+7.81%	+5.25%	+3.13%
GIN	+6.09%	+13.67%	+1.63%	+5.21%

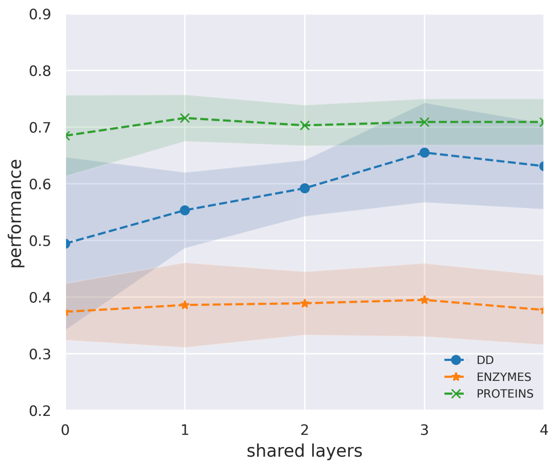
GT3 can also improve the classification performance in most cases, and more results can be found in Appendix A.2.

2.5.4 Ablation Study

In this subsection, we conduct an ablation study to understand the impact of key model components on the performance of the proposed GT3 including the SSL task and the adaptation constraint. This ablation study is conducted on DD, ENZYMES, and PROTEINS with the GCN model. Specifically, we study the effect of the adaptation constraint, global contrastive learning, and local contrastive learning by respectively eliminating each of them from GT3. These three variants of GT3 are denoted as GT3-w/o-constraint, GT3-w/o-global, and GT3-w/o-local. The performance is shown in Figure 2.7(a). We observe that the removal of each component will degrade the model's performance. This demonstrates the necessity of all three components.



(a) The Ablation Study of GT3.



(b) The Effect of Different Shared Layers.

Figure 2.7: (a)The Ablation Study of GT3 and (b) The Effect of Different Shared Layers.

2.5.5 The Effect of Shared Layers

We have also explored the effect of the number of shared layers on the performance of GPT3. To be specific, we vary the number of shared layers of the GCN model in GT3 as $\{0, 1, 2, 4\}$. Note that the model where the number of shared layers is zero denotes the model trained only by the main task. The graph classification performance on DD, ENZYMES, and PROTEINS is shown in Figure 2.7(b). From the figure, we can observe that most of the GT3 implementations with different shared layers can improve the classification performance over their baseline counterparts, while an appropriate choice of shared layers can boost the model performance significantly on some datasets such as DD.

2.6 Related Work

Test-Time Training. Test-time training with self-supervision is proposed in [101], which aims at improving the model generalization under distribution shift via solving a self-supervised task for test samples. This framework has empirically demonstrated its effectiveness in bridging the performance gap between the training set and test set in the image domain [6, 21, 101]. There are also works combining the test-time training and meta-learning learning [6], and attempting to apply test-time training in reinforcement learning [28]. Apart from these applied researches, a recent work tries to explore when test-time training thrives from a theoretical perspective [63].

Graph Neural Networks. Graph Neural Network is a successful generalization of Deep Neural Network over graph data. It has been empirically and theoretically proven to be very powerful in graph representation learning [120], and has been widely used in various applications, such as recommendation [35, 19], social network analysis [102] and natural language processing [43, 11]. GNNs typically refine node representations by feature aggregation and transformation. Most existent GNN models can be summarized into a message-passing framework consisting of message passing and feature update [24]. There is also another perspective unifying many GNN models as methods to solve a graph signal denoising problem [67]. Overall, GNN is an active research field and there are a lot of works focusing on improving the performance, robustness, and scalability of the GNN models [12, 46, 62].

2.7 Limitations

To the best of our knowledge, GT3 is the first test-time training framework specially designed for GNNs. While we have carefully designed each component of GT3, there still exist some limitations in the proposed framework at its current stage. First, it is very challenging to determine when to stop the model adaptation process during the test time. In other words, how to select the optimal model during the test-time training process is very hard, since there is no validation set. Although we have tried some heuristic methods to define the stopping criterion for model adaptation during the test-time training process, none of them work effectively on all the test samples from different datasets. After the empirical study, we end up with a simple stopping criterion, that is, to cease the model adaptation process after a fixed number of training steps. However, this criterion is very crude and may fail to find an optimal model for some test samples, especially when the test samples are very diverse. Second, although we have theoretically illustrated that the test-time training framework can be beneficial for GNNs on the graph classification task, we have not provided a principled method with theoretical guarantees, to develop self-supervised learning tasks that can benefit different main tasks in the graph test-time training framework. In other words, for different graph-related tasks, we need to design new self-supervised learning tasks for the test-time training based on intuitions and some preliminary study.

2.8 Conclusion

In this work, we design a test-time training framework for GNNs (GT3) for the graph classification tasks, with the aim to bridge the performance gap between the training set and the test set when there is a data distribution shift. As the first work to combine test-time training and GNNs, we empirically explore the rationality of the test-time training framework over GNNs via a layer-wise representation comparison of GNNs for different tasks. In addition, we validate that the GNN performance on graph classification can be benefited from test-time training from a theoretical perspective. Furthermore, we demonstrate the effectiveness of the proposed GT3 on extensive experiments and understand the impact of the model components via ablation study. As future work, it is worth studying test-time training for GNNs for other tasks on graphs such as node classification.

Also, it is interesting to explore how to automatically determine the number of shared layers and choose appropriate SSL tasks for various main tasks on different datasets.

CHAPTER 3

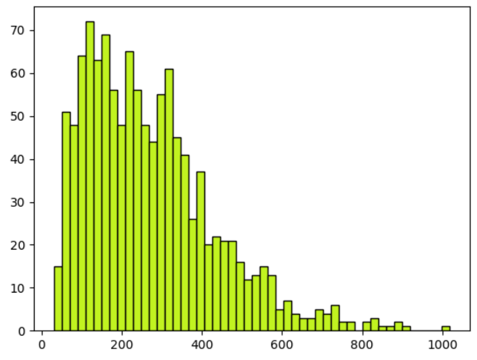
CUSTOMIZED GRAPH NEURAL NETWORKS

Graph Neural Networks (GNNs) have greatly advanced the task of graph classification. Typically, we first build a unified GNN model with graphs in a given training set and then use this unified model to predict labels of all the unseen graphs in the test set. However, graphs in the same dataset often have dramatically distinct structures, which indicates that a unified model may be sub-optimal given an individual graph. In this chapter, different from the training perspective proposed in Chapter 2, we propose to develop customized graph neural networks for graph classification from a novel learning perspective. Specifically, we propose a novel customized graph neural network framework, i.e., Customized-GNN. Given a graph sample, Customized-GNN can generate a sample-specific model for this graph based on its structure. Meanwhile, the proposed framework is very general that can be applied to numerous existing graph neural network models. Comprehensive experiments on various graph classification benchmarks demonstrate the effectiveness of the proposed framework.

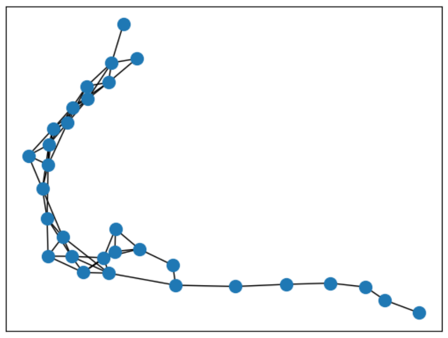
3.1 Introduction

Graphs are natural representations for many real-world data such as social networks [27, 52, 122], biological networks [9, 95] and chemical molecules [24, 91]. A crucial step to perform downstream tasks on graph data is to learn better representations. Deep neural networks have demonstrated great capabilities in representation learning for Euclidean data and thus have advanced numerous fields including speech recognition [77], computer vision [33], and natural language processing [15]. However, they cannot be directly applied to graph-structured data since graphs have complex topological structures. Recently, graph neural networks (GNNs) have generalized deep neural networks to graph data. GNNs typically update node representations by transforming, propagating, and aggregating node features across the graph. They have boosted the performance of many graph-related tasks such as node classification [52, 27], link prediction [22, 105, 134], and graph classification [69, 58, 23, 126].

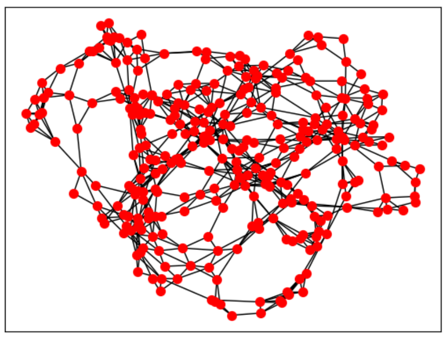
Graph classification is one of the most important and prevalent graph-related tasks [18], and in



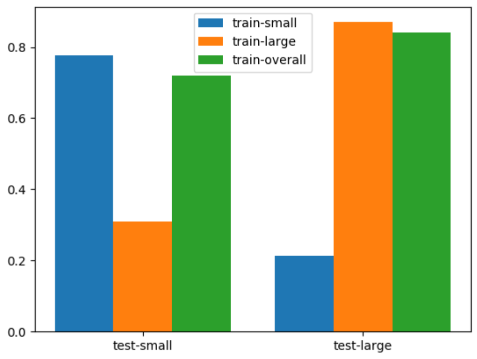
(a) Node size distribution.



(b) A graph with 31 nodes.



(c) A graph with 302 nodes.



(d) Classification accuracy.

Figure 3.1: An illustrative example of varied structural information and its impact on the performance of graph neural network-based graph classification. (a) demonstrates the distribution of graph size (i.e., the number of nodes) for protein graphs in the D&D dataset, where the graph size varies dramatically from 30 to 5, 748. ; (b) and (c) show two graphs sampled from the D&D dataset, which present very different structural properties; (d) shows classification performance of three models trained on training sets with various graph sizes.

this work, we aim to advance graph neural networks for the graph classification task. There are numerous real-world applications for graph classification. For example, proteins can be denoted as graphs [16], and the task to infer whether a protein functions as an enzyme or not can be regarded as a graph classification task; and it can also be applied to forecast Alzheimer's disease progression in which individual brains are represented as graphs [97]. Unlike data samples in classification tasks in other domains such as computer vision [89] and natural language processing [55], graph samples in the graph classification task are described not only by the input (node) features but their graph structures. Both the input node features and the graph structures play crucial roles in the graph classification tasks [126, 69, 23]. In reality, graphs in the same data set can present significantly different structural properties. Figure 3.1(a) demonstrates the distribution of graph size (i.e., the number of nodes) for protein graphs in the D&D dataset [16], where the graph size varies dramatically from 30 to $5,748$. We further illustrate two graphs sampled from the D&D dataset in Figures 3.1(b) and 3.1(c), respectively. These two graphs present very different structural properties such as the number of nodes, graph shapes, and diameters.

The above investigations indicate that graphs in the same data set could have dramatically distinct structural properties. It naturally raises the question – whether we should treat these graphs differently? To investigate this question, we take the graph size as the representative structure property and demonstrate how it affects the graph classification performance. Specifically, we divide graphs from D&D into two groups based on their graph size – one for graphs with a small number of nodes and the other for graphs with a large number of nodes. Then, we split each group into a training set and a test set. Next, we train three GNN models [1] based on three training sets – the small one, the large one, and the overall (a combination of the small and large one), separately. Then, we test their performance on the two test sets. The results are shown in Figure 3.1(d). In the test set with small/large graph sizes, the model trained on the training set with the same graph size can significantly outperform the other two models. Investigations and consistent observations on more settings can be found in Section 3.2.

These investigations suggest that a unified model is not optimal for graphs with diverse structure properties and efforts are desired to consider the structure-property difference among graphs. Hence, in this paper, we aim to learn "customized models" for graphs with different structural properties. A natural way is to divide the dataset into different splits according to structure properties and train a model for each split. However, we face enormous challenges to achieve this goal in practice. First, there are potentially several structure properties (graph size, density, etc.) affecting the performance, and we have no explicit knowledge about how the graphs should be split according to these properties.

[1]The GNN model for graph classification uses GCN [52] as the filtering operation and max-pooling as the pooling operation.

Second, dividing the dataset leads to small training sets for the splits, which may not be sufficient to train satisfactory models. To address these challenges, we propose a novel graph neural network framework, Customized-GNN, for graph classification. The Customized-GNN framework is trained on *all graphs in the given training set* (without splitting) and is able to produce customized GNN models for each individual graph. Specifically, we design an adaptor, which is able to smoothly adjust a general GNN model to a specific one according to the structural properties of a given graph. The general GNN model and the adaptor are learned during the training stage simultaneously utilizing all graphs.

Our major contributions are listed as follows: 1) We empirically observed that graphs in a given dataset could have dramatically distinct structural properties. Furthermore, it is not optimal to train a unified model for graphs with various structure properties for a graph classification task; 2) We propose a framework, Customized-GNN, which is able to generate a customized GNN model for each graph sample based on its structural properties. The proposed framework is general and can be directly applied to many existing graph neural network models; 3) We designed and conducted comprehensive experiments on numerous graph datasets from various domains to verify the effectiveness of the proposed framework.

3.2 Preliminary Data Analysis

In Figure 3.1, we have demonstrated that graphs in D&D have varied properties, which affected the performance of GNNs for graph classification. In this section, we aim to further investigate this phenomenon by answering the following two questions – (1) can the observations on D&D be extended to other datasets? and (2) whether incorporating these properties into the models can facilitate the performance?

We choose four representative graph datasets from different domains for this study including **D&D** [16], **ENZ** [95], **PROT** [9] and **RE-BI** [122]. We checked the properties such as node size and edge size. Similar to D&D, graphs in all datasets present very diverse properties. More details about these datasets can be found in Appendix B.1. Following the same setting as D&D, we divide each data into two groups according to the node size, i.e., large training and test (denoted as

"L-training" and "L-test") and small training and test (indicated as "S-training" and "S-test"). The results are demonstrated in Table 3.1. From the table, we make consistent observations with these in D&D – models trained on one property group (e.g., L-training) cannot perform well on the other property group (e.g., S-test).

Table 3.1: Graph classification accuracy on different node-size sets.

Accuracy (%)	**DD**		**ENZ**		**PROT**		**RE-BI**	
	S-test	L-test	S-test	L-test	S-test	L-test	S-test	L-test
S-training	66.2	55.7	45.0	20.0	76.5	51.4	88.6	46.6
L-training	47.2	77.8	27.0	39.0	45.6	78.6	29.3	83.1

To answer the second question, we divide D&D into several subsets based on the node size and then divide each subset into a sub-training set (80%) and a sub-test set (20%). We train models on different sub-training sets separately and then test their performance on all the sub-test sets. Specifically, we have trained four models on four different training sets from D&D, which are *training 1*, *training 2*, *training 3* containing graph samples with node sizes from different ranges, and *training* which is the combination of *training 1*, *2* and *3*. Then, we test four models on four test sets, i.e., *test 1*, *test 2*, *test 3* and *test* which is the combination of *test 1*, *2* and *3*. Statistics about these training sets are summarized in Table 3.2.

The performance of four models on the test sets is illustrated in Table 3.2. We note that the model trained on a specific training set performs much better on the corresponding test set that shares the same node size range than the other test sets. This suggests the potential to incorporate the structure properties into the model training. In addition, though *training 1*, *training 2*, and *training 3* have much fewer training samples, the models trained on specific training sets can achieve better performance on the corresponding test sets compared to these trained from the entire training set (or *training*). This indicates that a unified graph neural network that is trained from the entire training set is not optimal for graphs with various structure properties in the test set.

Discussion. Via the preliminary data analysis, we have established: (1) graphs in real-world data present distinct structure properties that tend to impact the graph classification performance of GNNs; and (2) incorporating the difference has the potential to boost the graph classification

Table 3.2: The classification accuracy of the models trained from four training sets in the D&D dataset and Statistics for four training sets.

Training set	Node size range	#Graphs	Accuracy(%)			
			test 1	test 2	test 3	test
training 1	[0,200]	369	76.1	41.2	26.7	50.9
training 2	[200,400]	392	43.5	75.3	82.2	62.4
training 3	[400,2000]	180	23.9	75.3	88.9	56.8
training	[0,2000]	941	64.1	61.9	75.6	66.2

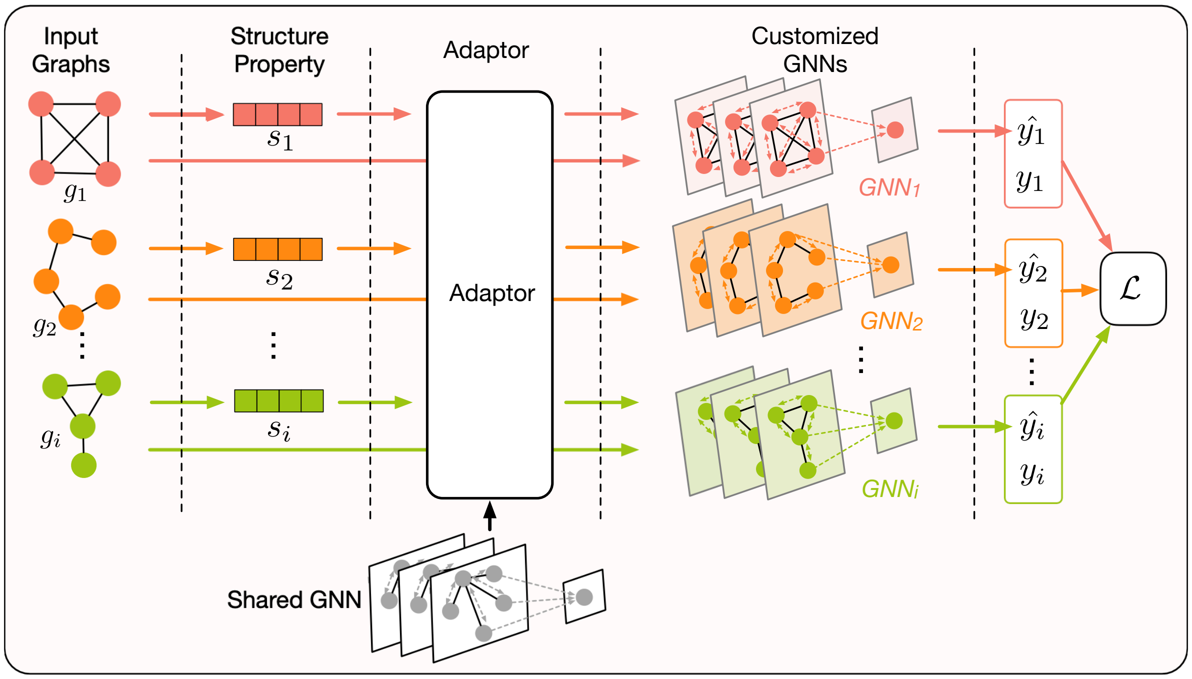


Figure 3.2: An overview of the proposed customized graph neural networks. Given a set of graph samples, the adaptor networks take their structure properties as input and generate corresponding adaptor parameters, which are used to adapt a shared GNN model (this could be any GNN model that works for the graph classification task) to a model specific for each graph sample. Each adapted model incorporates the structure information of a graph, and thus, is customized for this graph sample to make label predictions. With the predicted label and the ground truth label, we can calculate the overall loss, which is used to guide the optimization of the adaptor networks and the shared GNN model.

performance. These observations lay the foundations of the model design in the next section.

3.3 The Proposed Framework

In this section, we introduce the proposed framework Customized-GNN that has been designed for graphs with inherently distinct structure properties.

3.3.1 The Overall Design

As mentioned in earlier sections, graphs in real-world data inherently present distinct structural properties. Thus, we are desired to build distinct GNN models for them. To achieve this goal, we face tremendous challenges. First, we have no explicit knowledge about how the graph structure properties will influence graph neural network models. Second, if we separately train different models for graphs with different structure properties, we have to split the training sets for each model; as a consequence, the training data for each model could be very limited. For example, in the extreme case where each graph has unique graph structural properties, we only have one training sample for the corresponding model. Third, even if we can well train distinct GNN models for different graphs, during the test stage, for an unlabelled graph with unseen structural properties, it is hard to decide which trained model we should adopt to make the prediction. In this work, we propose a customized graph neural network framework, i.e., Customized-GNN, which can tackle the aforementioned challenges simultaneously.

An overview of the architecture of Customized-GNN is demonstrated in Figure 3.2. The basic idea of Customized-GNN is – it generates customized adaptor parameters for each graph sample g_i via an adaptor network with the graph structure properties as input. These generated adaptor parameters are used to adapt a shared GNN model denoted as GNN (this could be any GNN model that works for the graph classification task) to a model specific for the graph sample g_i. The adapted model GNN_i incorporates the structure information of graph g_i, and thus, is customized for the graph sample g_i.

With the proposed Customized-GNN framework, the first challenge is handled, since the influence is implicitly modeled by the adaptor networks, which can customize the shared GNN model to a graph sample-specific one. Furthermore, Customized-GNN can be trained on the entire training set without splitting it according to the graphs' structure properties. This not only solves the second challenge but also ensures that the trained model can preserve common knowledge from the entire training set. The third challenge is also automatically addressed by the Customized-GNN framework. Given an unseen graph g_j, the Customized-GNN framework first takes its graph structure information as input and generates adaptor parameters. Then, these generated adaptor parameters can be used to customize the general GNN model to a customized one GNN_j to predict the label of g_j.

Next, we introduce details about the adaptor network, the process of adapting a shared model to a specific one for a given graph, and the time complexity analysis of the proposed framework.

3.3.2 The Adaptor Network

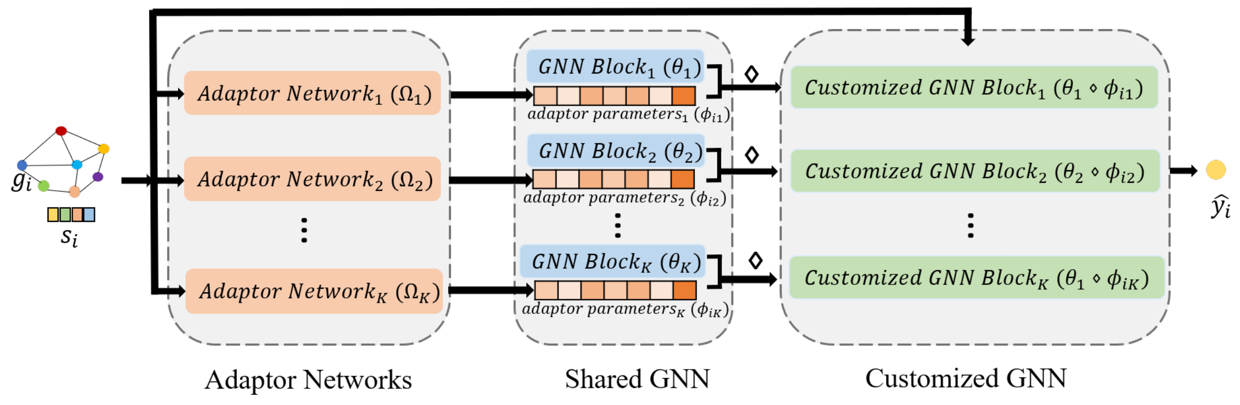
The goal of the adaptor network is to generate the adaptor parameters for a given graph. From the preliminary data analysis, we have the intuition that the customized GNN model for a specific graph sample should be correlated to its structural properties. However, there is no explicit knowledge about how these structural properties influence graph neural network models. To model this implicit mapping function, we propose to utilize a powerful neural network to generate the model adaptor parameters from the observed structure information of a given sample.

In addition, graph neural networks often consist of several subsequent filtering and pooling layers, which can be viewed as different GNN blocks. For example, K GNN blocks are shown in Figure 3.3. The graph structure properties of a given sample may have different influences on different GNN blocks. Hence, for each GNN block, we introduce one adaptor network to generate adaptor parameters for each block.

Specifically, we first extract a vector $\mathbf{s}_i$ to denote the structure information of a given graph g_i. We will discuss more details about $\mathbf{s}_i$ in the experiment section. As shown in Figure 3.3, the adaptor networks take the structure information $\mathbf{s}_i$ as input and generate the adaptation parameters for each block. In the case where there are K blocks in the graph neural network, we have K independent adaptor networks corresponding to the K blocks. Note that these adaptor networks share the same input $\mathbf{s}_i$ while their outputs are different. Specifically, the adaptor network for the j-th block can be expressed as follows:

$$\boldsymbol{\phi}_{ij} = h_j(\boldsymbol{s}_i; \Omega_j), j = 1, \ldots, K, \tag{3.1}$$

where Ω_j denotes the parameters of the j-th adaptor network and $\boldsymbol{\phi}_{ij}$ denotes its output, which will be used to adapt the j-th learning block. The adaptor network h_j can be modeled using any function.



Adaptor Network$_1$ (Ω_1)
Adaptor Network$_2$ (Ω_2)
Adaptor Network$_K$ (Ω_K)
GNN Block$_1$ (θ_1)
adaptor parameters$_1$ (ϕ_{i1})
GNN Block$_2$ (θ_2)
adaptor parameters$_2$ (ϕ_{i2})
GNN Block$_K$ (θ_K)
adaptor parameters$_K$ (ϕ_{iK})
Customized GNN Block$_1$ ($\theta_1 \diamond \phi_{i1}$)
Customized GNN Block$_2$ ($\theta_2 \diamond \phi_{i2}$)
Customized GNN Block$_K$ ($\theta_1 \diamond \phi_{iK}$)
g_i
s_i
$\hat{y}_i$
Adaptor Networks
Shared GNN
Customized GNN

Figure 3.3: An overview of the GNN adaptation process. Given a specific graph sample g_i, the adaptor networks consisting of K blocks take its graph structure properties s_i as input, and generate corresponding adaptor parameters to adapt each shared GNN block to get a customized GNN for g_i. This customized GNN is then used to make a prediction for g_i.

In this work, we utilize feed-forward neural networks due to their strong capability. According to the universal approximation theorem [38], a feed-forward neural the network can approximate any nonlinear functions. For convenience, we summarize the process of the K adaptor networks with s_i as input below:

$$\Phi_i = H(\mathbf{s}_i; \mathbf{\Omega}_H), \tag{3.2}$$

where Φ_i contains the generated adaptation parameters of all the GNN blocks for graph g_i and $\mathbf{\Omega}_H$ denotes the parameters of the K adaptor networks.

3.3.3 The Adapted Graph Neural Network

Any existing graph neural network model can be adapted by the Customized-GNN framework to generate sample-specific models based on the structure information. Therefore, we first generally introduce the GNN model for graph classification and describe how to adapt it given a specific sample. Then, we illustrate how to adapt specific GNN models.

3.3.3.1 A General Adapted Framework

A typical GNN framework for graph classification contains two types of layers, i.e., the filtering layer and the pooling layer. The filtering layer takes the graph structure and node representations as input and generates refined node representations as output. The pooling layer takes graph structure and node representations as input to produce a coarsened graph with a new graph and new node representations. A general GNN framework for graph classification contains K_p pooling layers, each of which follows K_f stacking filtering layers. Hence, there are $K = K_p * K_f$ learning blocks in the GNN framework. A graph-level representation can be obtained from these layers that can be further utilized to perform the prediction. Given a graph sample g_j, we need to adapt each of the K layers according to its adaptor parameters generated from the adaptor network. Via this process, we can generate a GNN model GNN_j specific to g_j.

Without loss of generality, when introducing a filtering layer or a pooling layer, we use an adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ and node representations $\mathbf{X} \in \mathbb{R}^{n \times d}$ to denote the input of these layers where n is the number of nodes and d is the dimension of node features. Then, the operation of a filtering layer can be described as follows:

$$\mathbf{X}_{new} = f(\mathbf{A}, \mathbf{X}; \theta_f) \tag{3.3}$$

where θ_f denotes the parameters in the filtering layer and $\mathbf{X}_{new} \in \mathbb{R}^{n \times d_{new}}$ denotes the refined node representations with dimension d_{new} generated by the filtering layer. Assuming ϕ_f is the corresponding adaptor parameter for this filtering layer, we adapt the model parameter θ_f of this filtering layer as follows:

$$\theta_f^m = \theta_f \diamond \phi_f, \tag{3.4}$$

where θ_f^m is the adapted model parameter that has the same dimension as the original model parameter θ_f; and $\diamond$ is the adaptation operator. The adaption operator can have various designs, which can be determined according to the specific GNN model. We will provide the details of the adaptation operator when we introduce concrete examples in the following subsections. Then, with the adapted model parameters, we can define the adapted filtering layer as follows:

$$\mathbf{X}_{new} = f(\mathbf{A}, \mathbf{X}; \theta_f \diamond \phi_f). \tag{3.5}$$

On the other hand, the process of a pooling layer can be described as follows:

$$\mathbf{A}_{new}, \mathbf{X}_{new} = p(\mathbf{A}, \mathbf{X}; \theta_p), \tag{3.6}$$

where θ_p denotes the parameters of the pooling layer, $\mathbf{A}_{new} \in \mathbb{R}^{n_{new} \times n_{new}}$ with $n_{new} < n$ is the adjacency matrix for the newly generated coarsened graph and $\mathbf{X}_{new} \in \mathbb{R}^{n_{new} \times d_{new}}$ is the learned node representations for the coarsened graph. Similarly, we adapt the model parameters of the pooling layer as follows:

$$\theta_p^m = \theta_p \diamond \phi_p, \tag{3.7}$$

which leads to the following adapted pooling layer:

$$\mathbf{A}_{new}, \mathbf{X}_{new} = p(\mathbf{A}, \mathbf{X}; \theta_p \diamond \phi_p), \tag{3.8}$$

where ϕ_p is the adaptation parameters generated by the adaptor network for this pooling layer.

For convenience, we summarize a general GNN model as $GNN(\cdot|\Theta_{GNN})$, where Θ_{GNN} is the parameters in all GNN blocks(i.e., θ_f, θ_p in all filtering and pooling layers). Then, for a graph sample g_i, we can adapt the GNN model $GNN(\cdot|\Theta_{GNN})$ to a customized model for g_i denoted as $GNN(\cdot|\Theta_{GNN} \Diamond \Phi_i)$. Note that, as shown in Eq. (3.2), Φ_i contains adaptation parameters of all GNN blocks for a graph sample g_i. The adaptation operations in all GNN blocks (including filtering and pooling layers) are summarized in $\Theta_{GNN} \Diamond \Phi_i$. There are numerous GNN models designed for graph classification [22, 88, 69, 128]. The proposed framework can be applied to the majority of these models, i.e., these models all can serve as the $GNN(\cdot|\Theta_{GNN})$ model mentioned above. In this work, we focus on three representative GNN models including GCN [52], DiffPool [126], and gPool [22]. We would like to leave the investigations of other GNN models as one future work. Next, we will give details on how to adapt GCN and DiffPool since gPool follows a similar adaptation process.

3.3.3.2 Adapted GCN: Customized-GCN

Graph Convolutional Network (GCN) [52] is originally proposed for the semi-supervised node classification task. The filtering layer in GCN is defined as follows:

$$\mathbf{X}_{new} = f(\mathbf{A}, \mathbf{X}; \theta_f) = \sigma(\tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{X} \mathbf{W}), \tag{3.9}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ represents the adjacency matrix with self-loops, $\tilde{\mathbf{D}} = \sum_j \tilde{\mathbf{A}}_{ij}$ is the diagonal degree matrix of $\tilde{\mathbf{A}}$ and $\mathbf{W} \in \mathbb{R}^{d \times d_{new}}$ denotes the trainable weight matrix in filtering layer and $\sigma(\cdot)$ is a nonlinear activation function. With the adaptation parameter ϕ_f for this corresponding filtering layer, the adapted filtering layer can be represented as follows:

$$\mathbf{X}_{new} = f(\mathbf{A}, \mathbf{X}; \theta_f \diamond \phi_f) = \sigma(\tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{X}(\mathbf{W} \diamond \phi_f)). \tag{3.10}$$

Specifically, we adopt FiLM [81] as the adaption operator. In this case, the dimension of the adaptor parameter is $2d$, i.e., $\phi_f \in \mathbb{R}^{2d}$. We split ϕ_f into two parts $\gamma_f \in \mathbb{R}^d$ and $\beta_f \in \mathbb{R}^d$ and then the adaptation operation can be expressed as follows:

$$\mathbf{W} \diamond \phi_f = (\mathbf{W} \odot br(\gamma_f, d_{new})) + br(\beta_f, d_{new}), \tag{3.11}$$

where $br(\mathbf{a}, k)$ is a broadcasting function that repeats k times for the vector $\mathbf{a}$; hence, $br(\gamma_f, d_{new}) \in \mathbb{R}^{d \times d_{new}}$ and $br(\beta_f, d_{new}) \in \mathbb{R}^{d \times d_{new}}$ have the same shape as $\mathbf{W}$ and $\odot$ denotes the element-wise multiplication between two matrices.

To utilize GCN for graph classification, we introduce a node-wise max pooling layer to generate graph representation from the node representations as follows:

$$\mathbf{x}_G = p(\mathbf{A}, \mathbf{X}; \theta_p) = max(\mathbf{X}), \tag{3.12}$$

where $\mathbf{x}_G \in \mathbb{R}^{1 \times d_{new}}$ denotes the graph-level representation and $max()$ takes the maximum over all the nodes. Note that the max-pooling operation does not involve learnable parameters and thus no adaptation is needed for it. We refer to an adapted GCN framework as Customized-GCN.

3.3.3.3 Adapted diffpool: Customized-DiffPool

DiffPool is a hierarchical graph-level representation learning method for graph classification [126]. The filtering layer in DiffPool is the same as Eq. (3.9) and its corresponding adapted version is shown in Eq. (3.10). Its pooling layer is defined as follows:

$$\mathbf{S} = softmax(f_a(\mathbf{A}, \mathbf{X}; \theta_{f_a})), \tag{3.13}$$

$$\mathbf{X}_{new} = \mathbf{S}^T \mathbf{Z}, \tag{3.14}$$

$$\mathbf{A}_{new} = \mathbf{S}^T \mathbf{A} \mathbf{S}, \tag{3.15}$$

where f_a is a filtering layer embedded in the pooling layer, $\mathbf{S} \in \mathbb{R}^{n \times n_{new}}$ is a soft-assignment matrix, which softly assigns each node into a supernode to generate a coarsened graph. Specifically, the structure and the node representations for the coarsened graph are generated by Eq. (3.15) and Eq. (3.14) respectively, where $\mathbf{Z} \in \mathbb{R}^{n \times d_{new}}$ is the output of the filtering layers. To adapt the pooling layer, we only need to adapt Eq. (3.13), which follows the same way as introduced in Eq. (3.10) as it is also a filtering layer. We refer to the adapted diffpool model as Customized-DiffPool.

3.3.4 Training and Test via the Customized Framework

Given a graph sample g_i with the adjacency matrix $\mathbf{A}_i$, and the feature matrix $\mathbf{X}_i$, the Customized-GNN framework performs the classification task as follows:

$$\tilde{y}_i = GNN(\boldsymbol{A}_i, \mathbf{X}_i; \boldsymbol{\Theta}_{\mathcal{G}NN} \Diamond H(\boldsymbol{s}_i; \boldsymbol{\Omega}_H)). \tag{3.16}$$

During the training, we are given a set $\mathcal{G} = \{g_i, y_i\}$ of N graphs as training samples, where each graph g_i is associated with a ground truth label y_i. Then, the objective function of Customized-GNN can be represented as follows:

$$\min_{\boldsymbol{\Omega}_H, \boldsymbol{\theta}_{\mathcal{G}NN}} \sum_{i=1}^{N} \mathcal{L}(y_i, GNN(\boldsymbol{A}_i, \mathbf{X}_i; \boldsymbol{\Theta}_{\mathcal{G}NN} \Diamond H(\boldsymbol{s}_i; \boldsymbol{\Omega}_H))), \tag{3.17}$$

where N is the number of training samples and $\mathcal{L}$ is a loss function. In this work, we use Cross-Entropy as the loss function and adopt ADAM [50] to optimize the objective.

During the test phase, the label of a given sample g_ℓ can be inferred using (3.16). Specifically, the graph structure information $\mathbf{s}_\ell$ of the sample is first utilized as the input of the adaptor network $H(\cdot; \boldsymbol{\Omega})$ to identify its distribution information, which is then utilized to adapt the shared model parameter Θ_{GNN} to generate a sample-specific model GNN_ℓ. This sample-specific model finally performs the classification for this sample.

3.3.5 Time Complexity Analysis

In this subsection, we analyze the additional time required to calculate the adaptation parameters and perform the adaptation. Specifically, we use the FiLM adaptation operator, as an example of the adaptor network. For convenience, the dimension of the output node features in all layers is assumed to be the same d. The dimension of the output of the adaptation network ϕ_f is $2d$. Furthermore, we assume that the input of the adaptation network, i.e., the graph property information $\mathbf{s}_i$ is with dimension s. Then, the time complexity to generate the adaptation parameters for a single block using Eq. (3.1) is $O(2d \cdot s) = O(d \cdot s)$. Furthermore, the time required to adapt the parameters for a single block with Eq. (3.11) is $O(d^2)$. Hence, for graph neural networks with K learning blocks, the time complexity to calculate the adaptation parameters and perform the adaptation for all learning blocks is $O(K \cdot d \cdot s + K \cdot d^2)$. Note that, the time complexity of a single filtering operation in Eq. (3.9) is $O(m \cdot d + n \cdot d^2)$ where m denotes the number of edges while n is the number of nodes. Therefore, the total time complexity for K learning blocks without adaptation is $O(K \cdot m \cdot d + K \cdot n \cdot d^2)$. Furthermore, s is typically small (much smaller than m); hence, the additional time complexity introduced by the adaptation operation is rather small.

3.4 Experiments

In this section, we conducted comprehensive experiments to verify the effectiveness of the proposed Customized-GNN framework. We first describe the experimental settings. Then, we evaluate the performance of the framework by comparing the original GCN, DiffPool, and gPool with the adapted GCN, DiffPool, gPool models by the Customized-GNN framework. Next, we analyze the importance of different components in the adaptor operator. Finally, we conduct case studies to further facilitate our understanding of the proposed method.

3.4.1 Experimental Settings

We carried out graph classification tasks on seven datasets. More details about these datasets can be found in Appendix B.1. Next, we describe the baselines. In Section 3.3, we apply the proposed framework to adapt three graph neural networks models: a basic graph convolutional network (GCN) [52], and two SOTA graph classification models DiffPool [126] and gPool [22], respectively. The corresponding adapted versions are Customized-GCN, Customized-DiffPool, and Customized-gPool, respectively. *Our evaluation purpose is if the proposed framework can boost the performance of existing models by adapting them to their corresponding customized versions.*

Thus, (1) to validate the effectiveness of the proposed model, we compare Customized-GCN, Customized-DiffPool, Customized-gPool with GCN, DiffPool, and gPool; and (2) we have not chosen models in [88, 69, 128] as baselines here but the proposed framework can be directly applied to adapt them as well. Note that in this work, we construct a set of simple structural features $\mathbf{s}_i$ of g_i such as the number of nodes, the number of edges, and the graph density; however, it is flexible to include other complex features by the proposed framework. Furthermore, we create baselines to directly concatenate the graph structure properties $\mathbf{s}_i$ to the output graph embedding of the GCN, DiffPool, and gPool model. Correspondingly, we call these three methods as Concat-GCN, Concat-Diff, and Concat-gPool. In addition, we develop baseline methods, Multi-GCN, Multi-Diff, and Multi-gPool. They learn multiple graph convolutional networks for graph samples with different structural information. More details of these baselines can be found in Appendix B.2.

3.4.2 Graph Classification Performance Comparison

In this subsection, we first perform the comparison following the traditional setting. To further demonstrate the advantage of the proposed frameworks, we show their adaptability when the properties of test graphs are different from these of training graphs. Following the setting in [126], for each graph dataset, we randomly shuffle the dataset and then split 90% of the data as the training set and the remaining 10% as a test set. We train all the models on the training set and evaluate their performance on the test set with accuracy as the measure. We repeat this process with different data shuffling and initialization seeds for 4 times and report the average performance and standard variance. In terms of the implementation details, the GCN/Customized-GCN model consists of 3 filtering layers and a single max-poling layer; the hidden dimension of each filtering layer is 20, and ReLU [76] activation is applied after each filtering layer. For DiffPool/Customized-DiffPool and gPool/Customized-gPool, we set $K_p = 2$, $K_f = 3$ and the dimension of hidden filtering layer 20. We adopt fully-connected networks to implement the adaptor networks in the Customized-GNN frameworks. Its input dimension is the same as the dimension of the graph's structural information.

The results are shown in Table 3.3. We notice that the Concat- and the Multi-versions of the GNN models can, in some cases, achieve comparable or even better performance than their corresponding

Table 3.3: Comparisons of graph classification performance in terms of accuracy.

Accuracy (%)	Datasets						
	COLLAB	**ENZ**	**PROT**	**DD**	**RE-BI**	**RE-5K**	**NCI109**
GCN	67.9±1.4	50.4±3.0	77.0±2.3	79.3±5.3	82.6±4.9	50.7±1.3	74.9±2.7
Concat-GCN	68.4±1.4	52.5±5.1	78.4±1.9	77.6±3.2	80.7±3.5	50.7±1.0	75.6±1.2
Multi-GCN-2	68.3±1.4	47.0±1.8	79.5±1.3	77.1±2.4	80.6±3.5	50.3±1.9	74.2±1.9
Multi-GCN-3	67.0±1.4	44.6±5.4	79.9±2.2	76.7±3.5	77.5±7.3	48.5±2.1	75.8 ±1.5
Customized-GCN	71.3±1.0	55.4±4.6	78.8±3.2	79.6±3.9	91.5±1.6	53.3±1.3	76.7±1.1
DiffPool	70.6±1.2	57.9±2.5	78.6±2.5	81.5±4.1	89.6±1.1	56.2±1.1	77.5±0.7
Concat-Diff	70.7±0.7	60.0±1.7	77.7±2.5	81.3±2.9	91.1±1.7	54.9±1.4	78.0±0.5
Multi-Diff-2	70.7±0.6	56.3±1.3	80.0±1.2	79.3±2.9	89.9±2.5	53.7±0.6	76.8±0.7
Multi-Diff-3	70.8±1.1	52.5±0.8	80.9±1.7	80.6±2.3	88.8±0.7	53.4±2.4	78.5±1.2
Customized-DiffPool	73.6±0.5	57.9±7.2	78.6±2.9	80.6±2.6	95.1±1.6	55.8±1.1	78.2±0.9
gPool	69.4±2.2	53.8±3.2	77.3±3.0	78.9±5.5	88.9±1.6	51.3±0.6	77.1±1.2
Concat-gPool	69.7±0.5	57.1±1.8	79.1±2.2	78.0±2.6	88.5±1.3	50.9±2.2	76.3±0.7
Multi-gPool-2	69.0±1.9	50.8±5.1	79.7±1.0	79.5±2.7	84.0±3.2	49.3±2.3	73.5±2.1
Multi-gPool-3	68.9±1.6	46.2±3.2	80.6±0.8	80.0±3.6	83.1±4.5	48.9±1.8	75.2±1.9
Customized-gPool	72.3±1.0	62.9±3.6	80.6±1.6	80.0±3.1	91.1±0.7	53.3±1.4	76.5±1.9

original versions. This indicates that utilizing the graph structure properties has the potential to help improve the model performance. However, the performance of these variants is not so stable across different datasets, which means that these simple methods are not suitable for all datasets. For example, the Concat- versions may work well on datasets where the label is directly related to the graph structure properties but fail on those datasets where graph structure properties have a more implicit impact on the labels. On the other hand, the performance of the Multi-version of the GNN models is heavily dependent on how the data is split into different groups. It is not practical to find good splits manually. Furthermore, simply training different models for different graphs can lead to unsatisfactory performance because less training data is available for each model. In contrast, our proposed Customized models learn sample-wise adaptation, which automatically finds suitable models for different data samples according to their graph structure properties. Compared with the original GCN, DiffPool, and gPool, the corresponding Customized models achieve better performance in most of the datasets. This demonstrates that the sample-wise adaptation performed by the Customized-GNN framework can boost the performance of GNN frameworks.

Adaptability Study. To further show the adaptability of the proposed framework to new graphs with different structures, we order graphs according to their node sizes in non-decreasing order.

Table 3.4: Adaptability study (Note here Cust-X denotes Customized-X).

Accuracy(%)	Methods					
	Cust-GCN	GCN	Cust-DiffPool	DiffPool	Cust-gPool	gPool
ENZ	22.2	20.5	25.6	22.2	35.0	24.8
RE-BI	70.2	50.4	78.6	52.7	80.0	59.9

Then, we use the first 80% of the data as a training set and the remaining 20% as a test set. The purpose of this setting is to simulate the case where the structures of graphs in the test set are different from those in the training set. We only show the results on the **ENZ** and **RE-BI** datasets in Table 3.4, since observations from other datasets are consistent. We note that (1) GCN, DiffPool, and gPool cannot work properly in this setting; and (2) the customized frameworks perform much better under this setting. These results demonstrate the ability of the learned Customized-GNNs to adapt GNNs to graphs with new properties.

3.4.3 Ablation Study

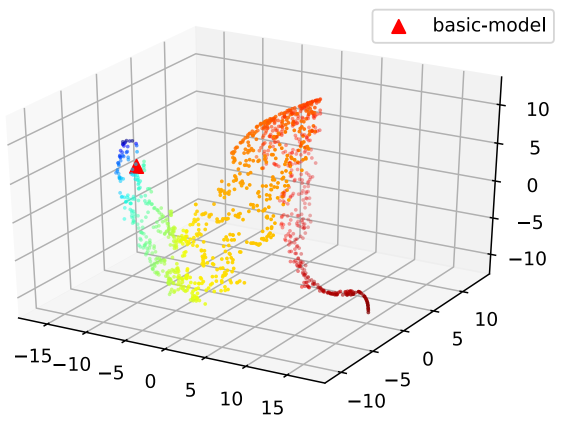
In this subsection, we investigate the effectiveness of different components in the adaptor operator in Eq. (3.11) used in our model. Specifically, we want to investigate whether γ_f and β_f play important roles in the adaptor operator by defining the variants of Customized-GCN – **Customized-GCN**$_\gamma$: It is a variant of the adaptor operator with only element-wise multiplication operation where instead of Eq. (3.11), the adaptation process is now expressed as $\mathbf{W} \diamond \phi_f = (\mathbf{W} \odot br(\gamma_f, d_{new}))$; and **Customized-GCN**$_\beta$: It is a variant of the adaptor operator with only element-wise addition operation where instead of Eq. (3.11), the adaptation process is now: $\mathbf{W} \diamond \phi_f = \mathbf{W} + br(\beta_f, d_{new})$.

Following the previous experimental setting, we compared Customized-GCN with its variants. The results are presented in Table 3.5. We observe that both **Customized-GCN**$_\gamma$ and **Customized-GCN**$_\beta$ can outperform the original GCN model. It indicates that both terms with γ and β are effective for the adaptation and utilizing either one of them can already adapt the original model in a reasonable manner. We also note that the Customized-GCN model outperforms both **Customized-GCN**$_\gamma$ and **Customized-GCN**$_\beta$ on most of the datasets. It demonstrates that the adaption effects of the term with γ and β are complementary to each other and combining them together can further enhance the performance.

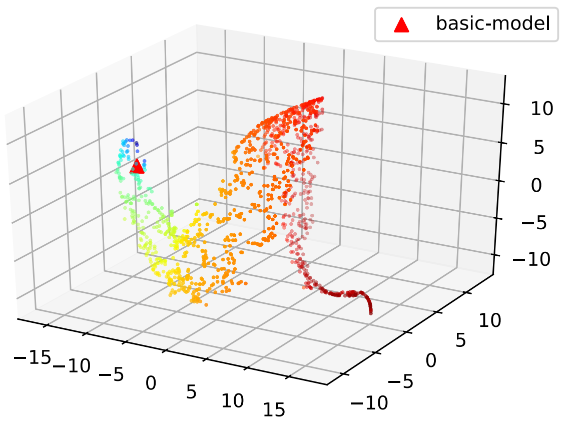
Table 3.5: Ablation study.

Accuracy (%)	Datasets						
	COLLAB	**ENZ**	**PROT**	**DD**	**RE-BI**	**RE-5K**	**NCI109**
GCN	69.9	51.8	76.6	77.2	81.9	50.4	75.7
Customized-GCN$_\gamma$	70.8	52.3	77.6	78.1	85.2	51.7	76.0
Customized-GCN$_\beta$	71.2	54.0	77.9	78.0	88.8	51.9	77.1
Customized-GCN	73.2	55.9	77.9	79.3	90.4	52.9	77.1

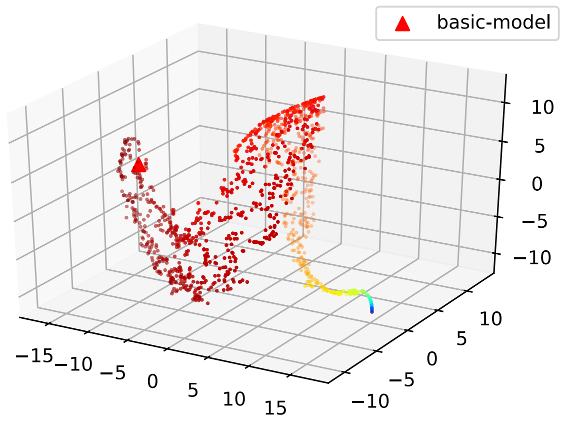
3.4.4 Case Study



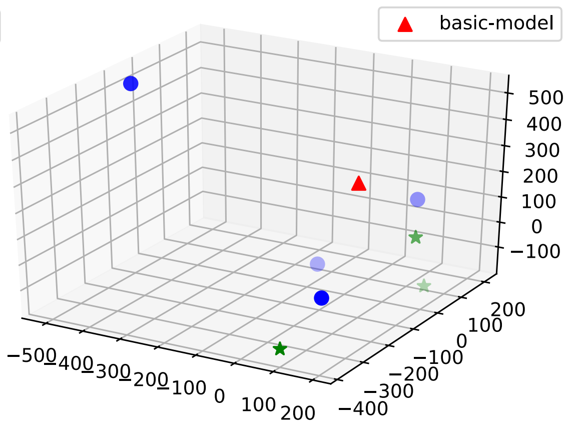
(a) Embeddings with node size.



(b) Embeddings with edge size.



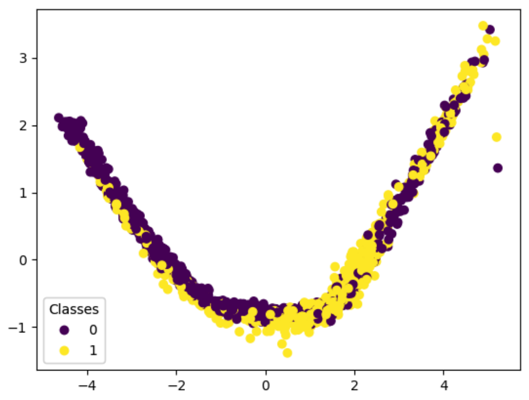
(c) Embeddings with density.



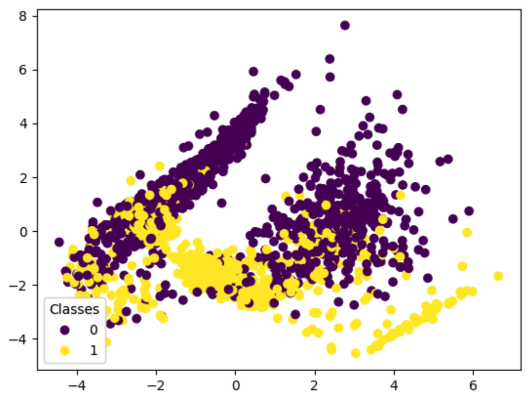
(d) Sample demonstration.

Figure 3.4: Case Study. (a) depicts the model embeddings and (b) demonstrates model embeddings for graphs that are mistakenly classified by GCN, but correctly classified by Customized GCN, and (c) and (d) illustrate the embeddings for graphs extracted by GCN and Customized-GCN, respectively.

To further illustrate the effectiveness of the proposed framework, we conducted case studies on D&D. First, we visualize the distribution of embeddings of sample-specific model parameters



(a) GCN.



(b) Customized-GCN.

Figure 3.5: Embedding Visualization.

for different graph samples with various node sizes, edge sizes, and densities. Specifically, we take the parameters of the first filtering layer of each sample-specific Customized-GCN framework and then utilize t-sne [70] to project these parameters to 3-dimensional embeddings. We visualize these 3-d embeddings in the form of scatter plot as shown in Figure 3.4(a), 3.4(b) and 3.4(c). Note that in these figures, the red triangle denotes the embedding of the parameters (i.e. $\mathbf{W}$) of the original GCN model (the one before adaptation). For each point in these figures, we use color to represent the scale of values in terms of node size (or edge size, density). Specifically, a deeper red color indicates a larger value, while a deeper blue color indicates a smaller value. We make some observations from Figure 3.4(a), 3.4(b) and 3.4(c). First, the proposed Customized-GCN framework indeed generates distinct models for different graph samples that are different from the original model. Second, the points with similar colors stay closely with each other, which means that graphs with similar structural information share similar models. In addition, in Figure 3.4(d), we illustrate the sample-specific model parameters for seven samples with a different number of nodes. They are misclassified by the original GCN model but correctly classified by the proposed Customized-GCN framework. It is obvious that Customized-GCN has generated seven different GCN models for these graph samples, each of which can successfully predict the label for the corresponding sample. We further visualize the graph embeddings before the classification layer, extracted by the model GCN and Customized-GCN. These embeddings from the two models are then projected to a 2-dimensional space via PCA and shown in Figure 3.5(a) and 3.5(b), respectively. We observe that the embeddings from different categories are better separated by Customized-GCN. This demonstrates that, compared to the original GCN, the proposed framework can get more distinct embeddings, and thus can achieve better classification performance.

3.5 Related Work

Graph Neural Networks have recently drawn great interest due to its strong representation capacity in graph-structured data in many real-world applications. Generally, graph neural networks can be divided into two categories: the spectral approaches and the non-spectral approaches. The spectral methods aim at defining the parameterized filters based on graph spectral theory by using graph Fourier transform and graph Laplacian [10, 13, 59, 52], and the non-spectral methods aim at defining parameterized filters based on nodes' spatial relations by aggregating information from neighboring nodes directly [27, 106].

Graph neural networks have advanced a wide variety of tasks including node classification [52, 27], link prediction [94, 134, 22] and graph classification [126, 69]. In the task of graph classification, one of the most important step is to get a good graph-level representation. A straight-forward way is to directly summarize the graph representation by globally combining the node representations [17]. Recently, there are some works investigating learning hierarchical graph representations by leveraging deterministic graph clustering algorithms [13, 20]. There also exist end-to-end models aiming at learning hierarchical graph representations, such as DiffPool [126]. MuchGNN [138] proposed to learn a set of graph channels at each layer to shrink the graph hierarchically. Furthermore, some methods [22, 57, 133] propose principles to select the most important k nodes to form a coarsened graph in each network layer. EigenPooling [69] is based on graph Fourier transform and is able to capture the local structural information. In [128], conditional random fields (CRF) are used to design the pooling operation.

3.6 Limitations

In this section, we discuss the limitations of the proposed framework. The proposed customized framework could suffer from scalability issue. The model size of the adaptor networks increases with the model size of the shared GNN, and the model size of the whole proposed framework would be rather large when the GNN model is large. This scalability issue has not become an obstacle for most existing GNN models, since the depth of these GNN models is often small currently. However, with the development of advanced research on GNNs, more and more complex GNN models have been designed, such as the GNN-based transformers, which makes the scalability issue of the customized GNN increasingly important.

3.7 Conclusion

In this paper, we propose a general graph neural network framework, Customized-GNN, to deal with graphs that have various graph structure properties. Comprehensive experiments demonstrated that the Customized-GNN framework can effectively adapt both flat and hierarchical GNNs to enhance their performance. Future research directions include better modeling the adaptor networks, considering more complex properties, and adapting more existing graph neural networks models.

CHAPTER 4

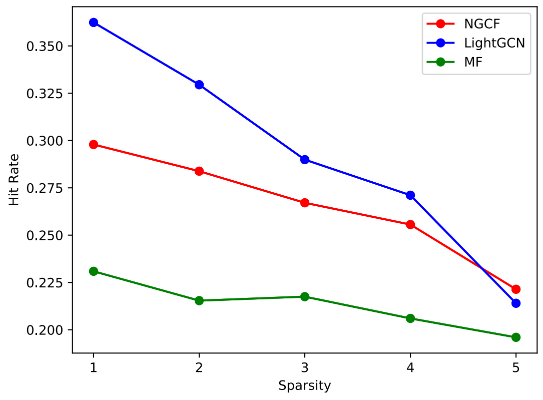
LOCALIZED GRAPH COLLABORATIVE FILTERING

User-item interactions in recommendations can be naturally denoted as a user-item bipartite graph. Given the success of graph neural networks (GNNs) in graph representation learning, GNN-based Collaborative Filtering (CF) methods have been proposed to advance recommender systems. These methods often make recommendations based on the learned user and item embeddings. However, we found that they do not perform well with sparse user-item graphs which are quite common in real-world recommendations. Therefore, in this chapter, we introduce a novel perspective to build GNN-based CF methods for recommendations which leads to the proposed framework of Localized Graph Collaborative Filtering (LGCF). One key advantage of LGCF is that it does not need to learn embeddings for each user and item, which is challenging in sparse scenarios. Alternatively, LGCF aims at encoding useful CF information into a localized graph and making recommendations based on such graph. Extensive experiments on various datasets validate the effectiveness of LGCF, especially in sparse scenarios. Furthermore, empirical results demonstrate that LGCF provides complementary information to the embedding-based CF model which can be utilized to boost recommendation performance.

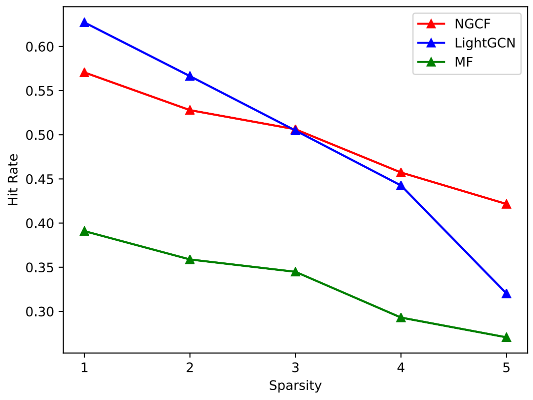
4.1 Introduction

The rapid development of information technology has facilitated an explosion of information, and it has accentuated the challenge of information overload. Recommender systems aim to mitigate the information overload problem by suggesting a small set of items for users to meet their personalized interests. They have been widely adopted by various online services such as e-commerce and social media [78]. The key to building a personalized recommender system lies in modeling users' preference on items based on their historical interactions (e.g., ratings and clicks), known as collaborative filtering (CF) [98]. In recommendation systems, the user-item interactions can naturally form a bipartite graph and GNNs have shown great potential to further improve the CF performance [110, 35]. GNNs are capable of capturing the CF signals encoded in the graph topology. Specifically, high-order user-item interactions can be explicitly captured by the stacked GNN layers, which contributes to learning expressive representations. For example, Neural Graph Collaborative Filtering (NGCF) [110] stacks several embedding propagation layers to explicitly encode the high-order connectivity in the user-item graph into the embedding process. LightGCN [35] simplifies the design for GNNs for recommendation by only including neighborhood aggregation while eliminating other operations such as feature transformation.

The key advantage of GNN-based CF methods is to learn both the embeddings and a GNN model by explicitly capturing the structural information in the user-item interaction graph. However, we empirically found that the improvement may no longer exist when the bipartite graph is very sparse. In Figure 4.1, we illustrate how the sparsity of the user-item bipartite graph affects the performance of representative CF models including MF, NGCF, and LightGCN. Specifically, we adjust the sparsity of the user-item graphs from two different commodity categories in the Tianchi dataset [2] by randomly removing edges from the original graph while not disrupting the graph connectivity. We use $x \in \{1, 2, 3, 4, 5\}$ (i.e., the x-axis) to denote the sparsity of a bipartite graph where the larger x is, the sparser the graph is. As shown in Figure 4.1, the performance of all the methods drops when the graph sparsity increases, in which GNN-based methods (NGCF and LightGCN) yield more dramatic performance reduction compared with MF. The inferior performance of GNN-based methods under the sparse setting could be attributed to the difficulty to learn high-quality user/item embeddings from limited user-item interactions. However, the user-item interaction graphs are often sparse in real-world recommendation scenarios [3]. Thus, in this paper, we study how to alleviate the challenge of sparsity while enjoying the merits of high-order topological connectivity. Note that in this work, we focus on addressing the recommendation tasks solely based on the historical interactions following the previous works [110, 35]. Our motivation lies in learning CF signals from local structures of the user-item interaction graph, and the proposed model is called Localized Graph Collaborative Filtering (LGCF). Particularly, given a user and an item, LGCF makes predictions about their interaction based on their local structural context in the bipartite graph, rather than the user and item embeddings. Further comparisons and discussions between the traditional GNN-based



(a) Tianchi-2798696.



(b) Tianchi-4527720.

Figure 4.1: The Performance of MF, NGCF, and LightGCN vs. the Graph Sparsity on Two Tianchi datasets.

CF methods and the proposed LGCF can be found in Appendix C.1.

To achieve the goal of LGCF, we face two main challenges: (1) how to construct the localized graph given two target nodes, and (2) how to capture the important CF information from the localized graph. To solve these two challenges, LGCF first samples a set of nodes for two target nodes from their neighborhoods and then generates a localized graph with this node set and their edges. Next, LGCF adopts a GNN model to extract a graph representation for each localized graph. To better capture the graph topological information, LGCF generates a label for each node according to their topological importance in the localized graph and takes the set of node labels as input attributes. Finally, LGCF makes predictions about the target user and item based on their corresponding localized graph representation.

The contributions of this paper are summarized as follows: (1) We study a new perspective to build GNN-based CF models for recommendations. Specifically, we aim at learning the recommendation-related patterns in the localized graphs induced from the input bipartite graph, instead of learning user and item embeddings; (2) We propose Localized Graph Collaborative Filtering (LGCF) which utilizes a GNN model and minimum distances-based node labeling function to generate graph representation for each user-item pair. In this way, we can encode the localized graph topological information and high-order connectivity into the graph representation simultaneously;

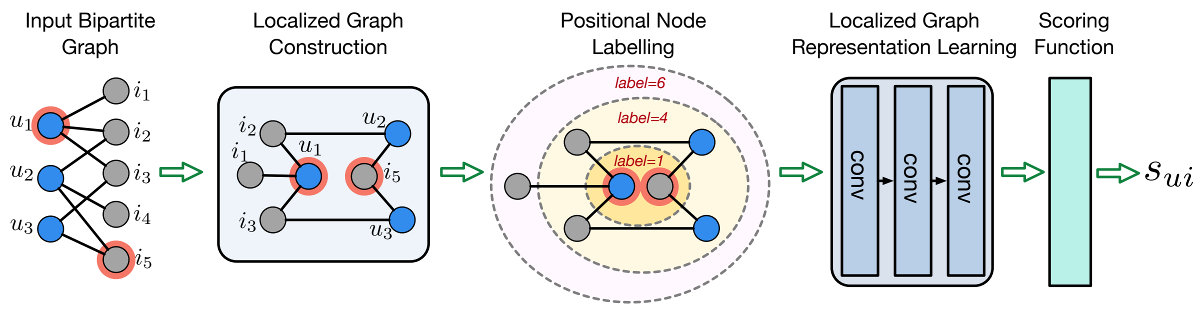

Input Bipartite Graph
Localized Graph Construction
Positional Node Labelling
Localized Graph Representation Learning
Scoring Function
label=6
label=4
label=1
conv
conv
conv
s_{ui}

Figure 4.2: An Overview of the Proposed Framework LGCF.

(3) We validate the effectiveness of LGCF on numerous real-world datasets. Especially, LGCF can achieve significantly better performance than representative GNN-based CF methods when user-item bipartite graphs are sparse.

4.2 Problem Statement

A CF-based recommendation dataset can be formulated as a bipartite graph $G = (\mathbf{U}, \mathbf{I}, \mathbf{E})$, in which $\mathbf{U} = \{u_1, u_2, ..., u_n\}$ denotes a set of n users, $\mathbf{I} = \{i_1, i_2, ..., i_m\}$ is the set of m items, and $\mathbf{E} = \{\mathbf{e_0}, \mathbf{e_1}, ..., \mathbf{e_l}\}$ (e.g., $e_j = (u_{e_j}, i_{e_j})$) indicates the edge set describing the historical interactions between users and items. Following previous works [110, 35], we assume that users and items have no content information. Given a candidate pair (u_j, i_k) consisting of a target user u_j and a potential item i_k, we need to calculate a preference score $s_{j,k}$ to indicate how likely this potential item should be recommended to the target user. To solve this problem, we aim at learning a desirable score mapping function $score()$, which generates a preference score $s_{jk} = score(u_j, i_k|\mathbf{E})$ to make recommendations based on the historical interactions $\mathbf{E}$.

4.3 The Proposed Framework

In this section, we first give an overall description of the proposed framework, then detail its key modules, next introduce the optimization objective and finally discuss integration strategies with existing GNN-based CF methods. Note that compared to existing GNN-based recommendation methods, the mode size of LGCF is much smaller and is independent of the number of users and items. More details about the mode size analysis can be found in Appendix C.2.

The goal of LGCF is to learn CF-related knowledge for each user-item pair from its local structures in an interaction bipartite graph. To achieve this goal, it provides solutions to tackle two aforementioned obstacles – a local structure extraction to construct the localized graph for the user-item pair and a powerful model to capture the CF-related patterns from the localized graph. An overview of LGCF is illustrated in Figure 4.2. Specifically, given the input bipartite graph, LGCF first constructs the localized graphs centered with the target user and the target item. To preserve edges with rich CF information in the localized graph, we develop a localized graph construction method to efficiently generate the localized graphs. After that, LGCF labels the nodes within the localized graphs according to the topological positions relative to the target nodes. Then a GNN-based graph representation learning module will embed the high-order connectivity along with the node positional annotations simultaneously. Through this process, LGCF eliminates the traditional embedding-based strategy and effectively captures the critical CF-related patterns. Finally, the scoring function module calculates the preference score based on the learned graph representation. To sum up, there are three key modules in this framework: (1) **the localized graph construction** module that extracts a localized graph covering most edges related to a given user-item pair; (2) **the localized graph representation learning module** that learns recommendation related representations from the localized graph; and (3) **the scoring function module** that computes a preference score based on the learned graph representation. Next, we will detail each module.

4.3.1 Localized Graph Construction

The localized graph construction module is designed to extract the localized graph covering the most important edges (i.e., collaborative filtering information) for a given user-item pair. We need to extract a localized graph for each user-item pair in the training process and the inference process, thus we aim to include the most representative edges for each pair with the consideration of scalability. To achieve these goals, we propose a localized graph construction as shown in Figure 4.3. This framework can be divided into three steps: (1) **step 1 -random walk**: we first take advantage of a random walk with restart (RWR) [104] method to sample neighboring nodes for the user node and the item node. By using RWR, we can sample neighboring nodes which are different hops away from the center node. As a consequence, we can get a representative localized graph that

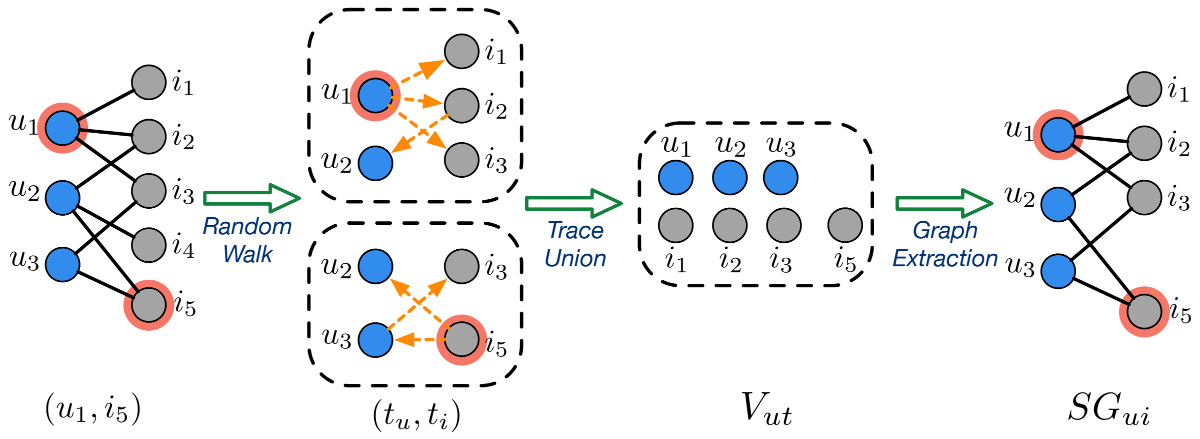


Figure 4.3: An Illustration of Localized Graph Extraction.

captures abundant context information. Meanwhile, RWR is a sampling method in essence, thus it can naturally improve the scalability of the proposed LGCF. Specifically, given a user-item bipartite graph G and a user-item pair (u,i), we perform a random walk on G from the user node u and the item node i, respectively. Starting from the original node, each walk travels to one of the connected nodes iteratively with a uniform probability. In addition, there is a pre-defined probability for each walk to return to the starting node at each step. In this way, more neighboring nodes can be included in the walk. Via RWR, we can get two traces t_u and t_i for the user node u and the item node i, respectively. Each trace then can be denoted as a node subset, i.e., V_u and V_i. (2) **step 2 - trace union**: Next, we merge V_u and V_i into a union node subset V_{ui}. (3) **step 3 - graph extraction**: With this union node subset V_{ui}, we can extract a localized graph from the original user-item graph. Specifically, all the nodes in V_{ui} and all the edges among these nodes are extracted from the original graph to build a localized graph, which is denoted as SG_{ui}. To conclude, SG_{ui} consists of neighboring nodes of different hops away from u and i and thus preserves important collaborative filtering information for the given user-item pair (u,i).

4.3.2 Localized Graph Representation Learning

This module aims at learning an overall representation of each user-item pair based on its extracted localized graph. We expect to encode the CF information and also the high-order connectivity related to a user-item pair into this representation. Therefore, we propose to use a graph neural network model since it can naturally capture the structural information and explicitly encode the high-order connectivity into the representation. Also, to further encode the topological information, we adopt a node labeling method to generate a positional label for each node based on its distance towards the user-item pair [129] where the node labels are considered as the input graph node attributes. Next, we will introduce the node labeling method and the adopted GNN model.

4.3.2.1 The Node Labeling Method

Following the common setting from the previous works [35, 110], we do not assume that node attributes are available for LGCF. Instead, we generate a label for each node to indicate its unique role in the localized graph. Specifically, there are three expectations for the generated label. First, it is important to distinguish the target user and item from other nodes through the generated node labels, as the model should be aware of the target user-item pair. Second, contextual nodes excluding the target user and item, play different roles in the recommendation prediction considering their different relative position towards the target user and item. Thus, we need to distinguish these nodes via the generated node labels based on their positions; Finally, for the consistency of node labels, the closer the node is to both the target user and target item, the more similar its label should be to that of the target ones. Therefore, we propose to use a node labeling method based on its distance to the user-item pair via Double-Radius Node Labeling (DRNL) [129] to generate a label for each node and take the generated node labels as the input node attribute for the GNN model. The key motivations of DRNL are to distinguish the target user node and the item node from other nodes while preserving their relatively positional relations. Specifically, we first assign label 1 to the target user node and the target item node to distinguish them from other nodes. Next, we assign labels to other nodes based on their minimum distances toward two target nodes on the extracted localized graph. Intuitively the closer the user or item node is to the target nodes, the more similar its label should be to that of the target nodes. For a specific node x on the graph, we evaluate its distance toward the target user and the target item by summing up its minimum distances to these two nodes. Since we set the label of the target user and the target item as 1, we will also assign these nearby nodes labels with small values. In particular, given nodes x and y, if the distance between x and the target nodes is smaller than that of y, the label value of x is supposed to be smaller than that of y. If the distances are the same, the node with a smaller minimum distance to the target user or the target item should have a label with a smaller value. DRNL adopts a hashing function $f_l()$ satisfying the criteria described above to compute node labels. The node labeling function $f_l(x)$ for each node x is summarized as follows:

$$f_l(x) = \begin{cases} 1, & \text{x=u or x=i,} \\ 1 + \min(d_u, d_i) + (d/2)^2, & \text{otherwise,} \end{cases} \tag{4.1}$$

where u and i denote the target user node and item node, respectively. d_u (d_i) represents the minimum distance between nodes x and u (i). $d = (d_u + d_i)$ is the sum of minimum distances which should be odd due to the nature of the bipartite graphs.

4.3.2.2 The GNN Model

For each user-item pair(u, i), we have extracted its localized graph SG_{ui}, and it can be denoted as $SG_{ui} = \{\mathbf{A}_{ui}, \mathbf{X}_{ui}\}$, where $\mathbf{A}_{ui}$ is the adjacency matrix of SG_{ui}, and $\mathbf{X}_{ui}$ is the input node attributes generated by above node labeling method. It is natural to take advantage of a GNN model to learn an overall representation for each pair based on its localized graph SG_{ui}, since GNNs have shown great potential in capturing complex patterns on graphs [120]. There are typically two key operations in a GNN model – the graph filtering operation to refine node representations and the graph pooling operation to abstract the graph-level representation from the node representations. In this work, we adopt graph convolutional filtering [52] as the graph filtering operation. Its process can be described as follows:

$$\mathbf{X}_{l+1} = \sigma(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}_l\mathbf{W}), \tag{4.2}$$

where $\mathbf{X}_l$ denotes the node representations in the l-th GNN layer, $\mathbf{W}$ is the transformation matrix to be learned, $\tilde{\mathbf{A}} = \mathbf{A}_{ui} + \mathbf{I}$ represents the adjacency matrix with self-loops, $\tilde{\mathbf{D}} = \sum_j \tilde{\mathbf{A}}_{jj}$ is the diagonal degree matrix of $\tilde{\mathbf{A}}$ and $\sigma()$ is the activation function. The node labels are set to be the input node representations $\mathbf{X}_0$. The graph pooling operation adopted by LGCF is sum pooling, which sums up representations all over the nodes for the graph-level representation. Its process can be denoted as follows:

$$\mathbf{x}_{ui} = pool(\mathbf{X}_L) = sum(\mathbf{X}_L), \tag{4.3}$$

where $\mathbf{x}_{ui}$ denotes the graph representation for the user-item pair (u, i) and $\mathbf{X}_L$ is the node representations outputted by the last layer. The overall process of the GNN model can be summarized as:

$$\mathbf{x}_{ui} = GNN(\mathbf{A}_{ui}, \mathbf{X}_0; \boldsymbol{\theta}_{GNN}), \tag{4.4}$$

where $\boldsymbol{\theta}_{GNN}$ denotes the parameters to be learned in the GNN model. It is straightforward to extend LGCF with other graph filtering operations [120, 106] and graph pooling operations [69, 22], and we will leave such investigation as one future work.

4.3.3 The Scoring Function

The scoring function aims at giving a score for a user-item pair (u, i) based on its corresponding localized graph representation $\mathbf{x}_{ui}$. The higher the score is, the more likely the item should be recommended to the user. We use a single-layer linear neural network to model the scoring function in LGCF, which can be described as follows:

$$s_{ui} = score(\mathbf{x}_{ui}) = \sigma(\mathbf{x}_{ui}^T * \mathbf{w}), \tag{4.5}$$

where $\mathbf{w}$ is the transformation vector to be learned and σ is the sigmoid function.

4.3.4 Model Optimization

In this work, we adopt the pairwise Bayesian Personalized Ranking (BPR) loss to train the framework. BPR loss is one of the most popular objective functions in recommendation tasks. It takes the relative order of the positive node pairs and negative node pairs where the positive node pairs are existent user-item node pairs observed in graphs, while the negative node pairs are non-existent user-item node pairs generated by negative sampling. Specifically, BPR assumes that the prediction scores of the positive node pairs should be larger than these of the corresponding negative ones. The objective function of our model is as follows:

$$\min_{\mathbf{\Theta}_{GNN},\boldsymbol{w}} \sum_{(u,i,i')\in O} -\ln\sigma(s_{ui} - s_{ui'}), \tag{4.6}$$

where $O = \{(u,i,i')|(u,i) \in E, (u,i') \in E^{-}\}$ denotes the training data for the graph G, E is the set of the existent interactions between users and items in G and E^{-} indicates the set of the non-existent interactions. In this work, we adopt ADAM [50] to optimize the objective.

4.3.5 Integration with Embedding-based CF Methods

LGCF aims at capturing the collaborative filtering information from a localized perspective by encoding the localized CF information into a subgraph-level representation, while embedding-based GNN methods such as NGCF and LightGCN are to learn a set of user and item embeddings in the latent space based on all the user-item interactions. Therefore, LGCF provides a new perspective to building GNN-based recommendations. As a consequence, LGCF is likely to provide complementary information to existing GNN-based methods and integrating it with existing methods has the potential to boost the recommendation performance. In this subsection, we discuss strategies to combine LGCF with existing GNN-based methods. In particular, we propose two strategies, i.e., *LGCF-emb* and *LGCF-ens*. Before we detail these strategies, we first denote the refined user u and item i embedding generated by embedding-based GNN methods as $\mathbf{h}_u$ and $\mathbf{h}_i$, and the representation outputted by LGCG for a user-item pair (u,i) as $\mathbf{x}_{ui}$. In this work, we focus on how to integrate LGCF with LightGCN given the competitive performance of LightGCN.

4.3.5.1 LGCF-emb

In LGCF-emb, we propose to directly concatenate the embeddings generated by LGCF and LightGCN, and then the scoring function takes the concatenated embedding as input to generate a score. Finally, the LGCF model and the LightGCN model are jointly trained by optimizing the objective function in Eq. 4.6. The concatenation process is summarized as follows:

$$\mathbf{h}_{ui} = (\mathbf{h}_u * \mathbf{h}_i)||\mathbf{x}_{ui}, \tag{4.7}$$

where $*$ denotes element-wise multiplication and $||$ represents the concatenation operation.

4.3.5.2 LGCF-ens

In LGCF-ens, we propose to combine LGCF and LightGCN in an ensemble way. Specifically, we conduct a weighted combination of the scores generated by LGCF and LightGCN. In addition, instead of joint training, we train LGCF and LightGCN, separately. The final score for a user-item pair (u, i) can be computed as follows:

$$s_{ui} = score(\mathbf{x}_{ui}) + \lambda \cdot (\mathbf{h}_u^{\mathrm{T}} \mathbf{h}_i), \tag{4.8}$$

where $score()$ denotes the scoring function in Eq 4.5 and λ can be either a predefined hyper-parameter or learnable parameter.

4.4 Experiment

In this section, we conduct extensive experiments on various real-world datasets to validate the effectiveness of the proposed LGCF. Via experiments, we aim to answer the following questions:

Q1: How does LGCF perform compared with the state-of-the-art CF models on the sparse setting?

Q2: How does LGCF perform compared with the state-of-the-art CF models on the normal setting?

Q3: Can LGCF be complementary to the embedding-based GNN CF models?

Q4: How does the sparsity of data affect the performance of LGCF and other CF models? In the following subsections, we will first introduce experimental settings, next design experiments to answer the above questions and we further probe to understand the working of LGCF.

4.4.1 Experimental Settings

In our experiments, we have tested the proposed framework on twelve datasets, which are from three different real-world recommendation scenarios: Tianchi[2], Amazon[71], and MovieLens[30]. Specifically, we construct seven datasets corresponding to seven-item categories from Tianchi and denote each of them as Tianchi-ID, where ID indicates the item category, two datasets from two item categories from Amazon, and two datasets from two item categories from MovieLens. More

Table 4.1: Densities of the Sparse User-item Bipartite Graphs. Here "Density" is computed specifically for the interaction graph as $Density = \#Edges \div (\#Users \times \#Items)$.

	Tianchi							**Amazon**		**MovieLens**	
	685988	**2798696**	**4527720**	**174490**	**61626**	**810632**	**3937919**	**Beauty**	**Gift**	**War**	**Romance**
density	0.0019	0.0106	0.0074	0.0036	0.0014	0.0019	0.0064	0.0121	0.0071	0.0113	0.0027

details about these datasets can be found in Appendix C.3. Note that it is common to extract a user-item interaction dataset based on the item category from a large recommendation dataset for evaluations that have been widely adopted by existing works [71, 110, 35].

We mainly compare the proposed method with LightGCN [35] and NGCF [110], which have empirically outperformed most recent CF-based models including NeuMF [36], PinSage [125] and HOP-Rec [123]. In addition, we also compare LGCF with one of the most classic factorization-based recommendation models – MF [53]. We use HR and NDCG as evaluation metrics in our experiments. More details about the baselines and evaluation details can be found in Appendix C.4.

4.4.2 Performance in Sparse Scenarios

The major motivation of LGCF is to improve the recommendation performance when there are a few user-item interactions or user-item interactions are sparse. Therefore, we start the evaluation by comparing the proposed method with baselines under the sparse setting, which correspondingly answers **Q1**. Specifically, to obtain a sparse training user-item interaction bipartite graph, we randomly select as many interactions as possible in the validation set or the test set while ensuring no isolated nodes in the training set and no cold-start nodes in the validation set and the test set. After this process, the densities of the resulting sparse graphs are summarized in Table 4.1. The performance comparison on these sparse datasets is demonstrated in Table 4.2. We make the following observations: 1) The performance of LightGCN is worse than NGCF on most datasets. These results suggest that only neighborhood aggregation may not help to embed refinement in sparse scenarios; 2) LGCF outperforms other baselines on the sparse datasets significantly, which supports that capturing local structures is more effective than learning user and item embeddings in sparse scenarios. Meanwhile, this validates the effectiveness of LGCF in capturing CF information from local structures on sparse user-item bipartite graphs. More details about how the sparsity

Table 4.2: Performance Comparison under the Sparse Setting.

HR(%)	Tianchi							Amazon		MovieLens	
	685988	**2798696**	**4527720**	**174490**	**61626**	**810632**	**3937919**	**Beauty**	**Gift**	**War**	**Romance**
MF	28.4±2.3	19.8±4.9	25.4±3.1	27.0±1.7	15.6±1.2	25.2±1.9	38.7±5.2	64.3±7.4	12.7±0.3	55.8±3.0	57.7±1.0
NGCF	36.7±3.0	25.6±4.2	32.7±3.3	37.2±4.2	17.8±2.2	36.0 ±2.3	47.5±3.0	83.1±0.7	14.4±0.3	63.5±1.1	71.0±0.2
LightGCN	29.4±2.1	22.1±3.9	25.9± 3.6	25.9±3.6	18.8±1.8	28.2±1.4	24.4±1.7	39.1±6.6	14.5±0.3	32.4±2.0	36.0±1.0
LGCF	65.3±0.4	39.7±0.6	60.2±1.0	66.2±0.5	41.6±3.0	60.7±0.2	61.6±0.2	88.0±0.4	32.4±2.0	81.1±0.5	82.7± 1.2

Table 4.3: Performance Comparison under the Normal Settings.

HR(%)	Tianchi							Amazon		MovieLens	
	685988	**2798696**	**4527720**	**174490**	**61626**	**810632**	**3937919**	**Beauty**	**Gift**	**War**	**Romance**
MF	57.3±3.0	22.0±3.0	43.9±4.1	41.6±3.3	22.3±4.6	70.7± 1.6	38.7±5.3	93.7±1.4	44.2±4.1	72.0±1.7	77.7±0.5
NGCF	69.6±1.8	29.7±3.7	56.0±2.1	55.2±3.0	37.1±3.7	53.6±2.9	47.5±3.0	93.5±1.1	46.4±2.4	75.5±0.5	82.6±0.2
LightGCN	76.2±1.2	32.4±2.9	62.7±1.7	60.3±1.1	44.0±3.5	74.1±0.6	55.6±3.6	94.4±1.1	51.4±0.3	80.9±1.0	85.3±0.2
LGCF	76.9±1.2	49.3±2.5	59.3±1.7	63.4±3.5	46.9±5.6	71.2±2.3	55.7±1.6	91.5±2.5	36.9±6.1	65.5±12.9	77.3±5.6
LGCF-emb	72.1±7.1	46.3±3.1	54.1±5.9	62.4±3.7	50.0±3.0	71.0±3.1	57.8±3.0	92.6±2.0	36.6±8.3	64.7±7.6	81.5±2.8
LGCF-ens	76.8±1.1	44.8±4.1	62.7±3.3	67.3 ±2.1	54.5±2.4	74.9±1.0	61.7±2.1	94.7±0.7	52.8±1.0	87.5±0.4	88.5±0.4

impacts performance and why LGCF can improve the performance will be provided in the following subsections.

4.4.3 Performance in Normal Scenarios

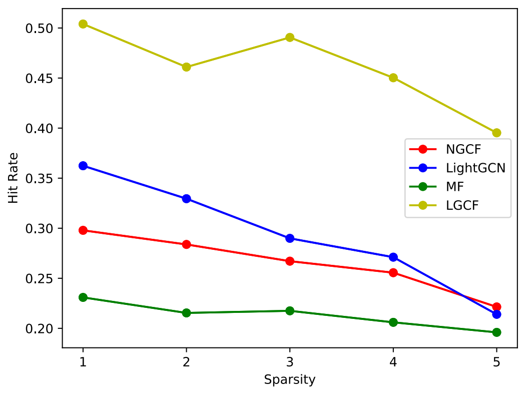
In the last subsection, we empirically demonstrated the superior of LGCF over the baseline methods in sparse scenarios. In this section, we further study how LGCF performs in normal settings where we ensure we have sufficient training data. To achieve the goal, we randomly select 90% of user-item interactions as the training set and then divide the remaining interactions into the validation set and the test set equally. Since LGCF makes recommendations from a different perspective from LGCF and LightGCN, we also investigate if LGCF is complementary to them. These experiments aim to answer **Q2** and **Q3**. The results are summarized in Table 4.3. It can be observed: 1) LGCF can often perform reasonably by achieving comparable performance with the baseline methods on most datasets; 2) LGCF-ens has shown significant performance improvement over both LGCF and LightGCN on most datasets. This demonstrates that LGCF indeed provides complementary information to embedding-based CF methods. Thus, there is great potential to integrate these methods for the recommendation task; 3) LGCF-emb does not perform well on some datasets. It directly concatenates embeddings from LGCF and LightGCN. However, the embeddings from LGCF and LightGCN could be not aligned. Among the integration methods, LGCF-ens achieves consistent and better performance compared to LGCF-emb.

4.4.4 The Impact of Sparsity

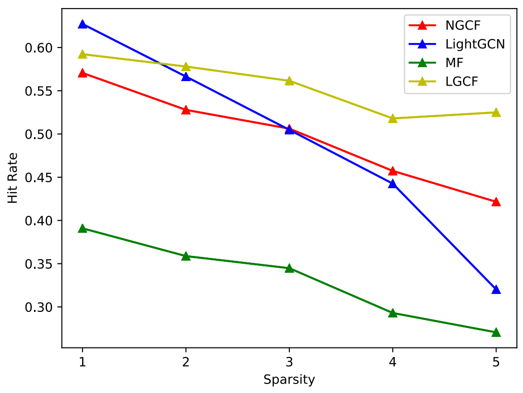
In this subsection, we further explore **Q4**: how the sparsity of the training user-item bipartite graph affects the performance of different models. Specifically, we first randomly select 10% user-item interactions and split them equally into the validation set and the test set. We constrain there are no cold-start users and items in the validation set and the test set. We take the remaining 90% interactions as the full training set and denote its sparsity as 1. Next, we divide the full training set into two sets: the necessary set and the additional set. The necessary set consists of the minimum interactions to ensure all the users and items are connected with at least one neighbor and the additional set includes the remaining interactions. We randomly select 20%, 40%, 60%, and 80% of interactions from the additional set and exclude them from the full training set to increase the sparsity of the training user-item bipartite graphs. Here we denote the sparsity of these training graphs as $2, 3, 4, 5$, respectively. The performance of LGCF, LGCF-enz, and the baseline methods are shown in Figure 4.4. (We have similar observations on most of the datasets, and we select four of them to report due to the space limitation) We can observe that the performance of all the methods tends to decrease with the increase of graph sparsity. However, the performance drop of LightGCN and NGCF is relatively more significant than that of the proposed LGCF. LightGCN and LGCF typically perform comparably on the least sparse cases, and then the performance gap tends to increase with the graph sparsity raises. In addition, LGCF-enz is more robust to the sparsity change compared to LightGCN, which demonstrates that the integration of LGCF can help LightGCN reduce sensitivity towards graph sparsity.

4.4.5 Further Probing

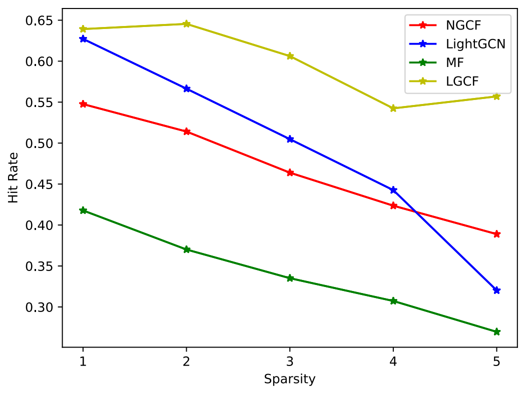
In this subsection, we explore to further understand how LGCF works. As mentioned before, LGCF provides a new perspective to building GNN-based CF methods. Experimental results in Table 4.2 have demonstrated that LGCF is complementary to LightGCN. Thus, we first examine how LGCF is complementary to LightGCN for different types of user-item pairs. Moreover, localized graphs play a crucial role in LGCF. Therefore, we investigate if localized graphs for positive and negative pairs are distinguishable via case studies.



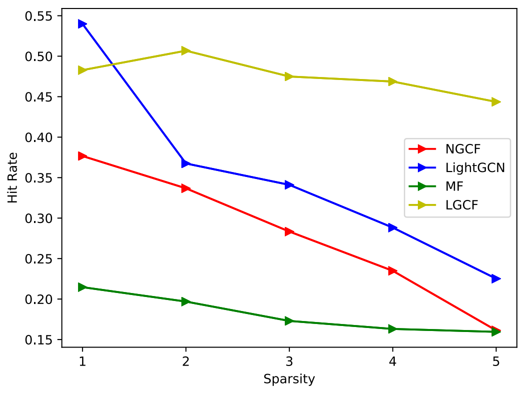
(a) Tianchi-2798696.



(b) Tianchi-4527720.



(c) Tianchi-174490.

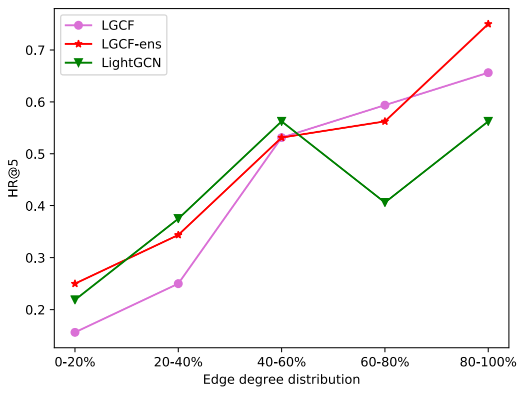


(d) Tianchi-61626.

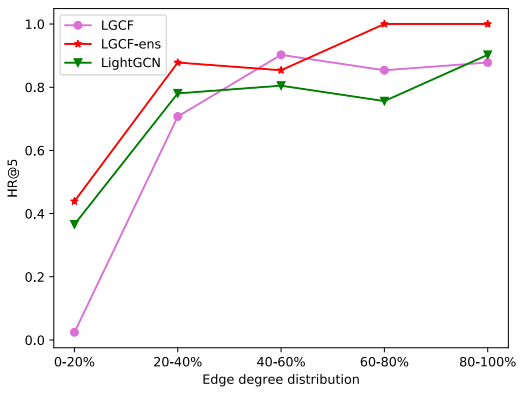
Figure 4.4: The Performance of Different Methods vs. the Graph Sparsity on Four Tianchi Datasets.

4.4.5.1 How is LGCF complementary to LightGCN

To explore what kind of user-item pairs can benefit from integrating LGCF and LightGCN, we divide the test set in normal scenarios into different groups based on the average degree of the user and the item for a user-item pair. Specifically, we first sort all the user-item pairs in the test set in ascending order according to their average degree, and divide the sorted pairs into 5 groups. We then train models on the same training set and test them on these groups. The performance of LightGCN, LGCF and LGCF-ens is shown in Figure 4.5. More results on different datasets can be found in Appendix C.5. We first note that the performance of all methods tends to increase when the degree increases. In most cases, for all degree groups, LGCF-ens (or integrating LGCF



(a) Tianchi-61626.



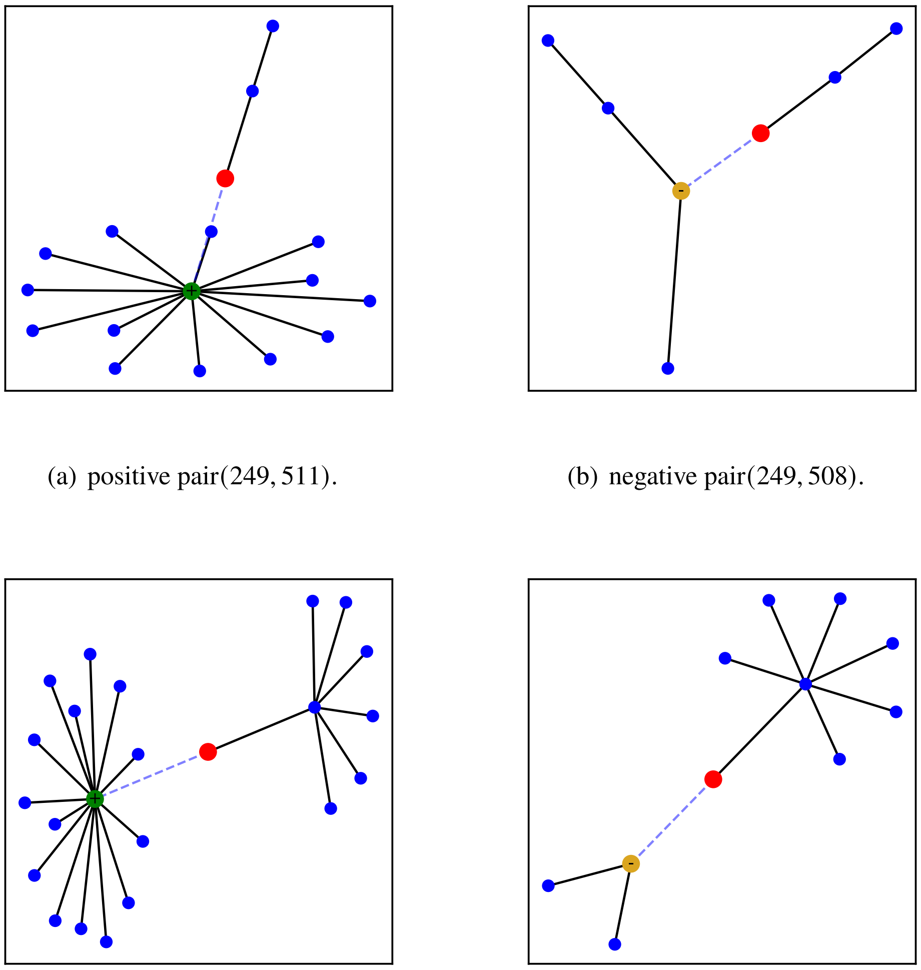
(b) MovieLens-War.

Figure 4.5: Performance Analysis with Degree.

and LightGCN) can boost the recommendation performance. This improvement is relatively more significant for groups with small degrees. These results suggest that LGCF provides complementary information to LightGCN for all degree groups, especially for these groups with small degrees. This property has significance in practice since how to make recommendations for user-item pairs with limited interactions is still a challenging problem in real-world recommender systems. In addition, this property makes the proposed method potentially computation-efficient for large-scale datasets, since we can first filter items that are less likely to be preferred by a given user based on embeddings and then leverage LGCF to rank the remaining items.

4.4.5.2 Localized Graphs

The key to the success of LGCF is that the extracted localized graphs for positive and negative pairs are distinguishable. In other words, they have distinct structures. Thus, we visualize the extracted localized graphs for two positive user-item pairs and their corresponding negative pairs in Figure 4.6. These two pairs are correctly predicted by LGCF, while wrongly predicted by LightGCN. For the figure, we observe that the localized graphs for positive pairs are substantially different from these of negative pairs, and thus LGCF can distinguish them and make correct predictions. These case studies not only validate the feasibility of using localized graphs for recommendations but also further demonstrate that recommendations via localized graphs are complementary to these via



(a) positive pair$(249, 511)$. (b) negative pair$(249, 508)$.

(c) positive pair$(87, 459)$. (d) negative pair$(87, 592)$.

Figure 4.6: Case Studies on Localized Graphs.

embeddings.

4.5 Related work

Collaborative Filtering (CF) is one of the most popular techniques for recommendation [5]. Its core idea is to make predictions of a user's preference by analyzing its historical interests. There are basically two types of CF methods: memory-based CF methods and model-based models. Memory-based CF methods aim at making predictions by memorizing similar users' (or items') historical interactions [37]. Model-based CF methods [53] aim at predicting by inferring from an underlying model. With the rapid development of deep neural networks (DNNs), there are more and more works exploring incorporating DNN techniques into model-based CF methods [36, 121]. As DNN successfully generalizes to graphs, graph neural networks (GNNs) have drawn great attention. User-item interactions in recommendations can be naturally denoted as a bipartite graph. Thus, more and more works study how to utilize GNN techniques in model-based CF methods including HOP-Rec [123], PinSage [125], NGCF [110] and LightGCN [35]. Specifically, HOP-Rec [123] includes multi-hop connections into the MF method. PinSage [125] is developed based on GraphSage [27], and it can be directly applied to industry-level recommendations. NGCF [110] proposes to explicitly encode high-order connectivity into representation via graph filtering operation. LightGCN [35] simplifies the design of GNN especially for the recommendation task by eliminating feature transformation and activation function.

4.6 Conclusion

In this paper, we propose a GNN-based CF model LGCF from a novel perspective for recommendations. It aims at encoding the collaborative filtering information into a localized graph representation. Different from existing embedding-based CF models, LGCF does not require learning embeddings for each user and item. Instead, it focuses on learning the recommendation-related patterns from the localized graph for each specific user-item pair. Extensive experiments have been conducted to demonstrate the effectiveness of LGCF in recommendation tasks. In the future, we will explore other graph filter and pooling operations to learn the localized graph representations. In addition, some advanced techniques such as the attention mechanisms can be applied to further incorporate different types of GNN-based CF models.

CHAPTER 5

AN ADAPTIVE GRAPH PRE-TRAINING FRAMEWORK FOR LOCALIZED COLLABORATIVE FILTERING

Graph neural networks (GNNs) have been widely applied in recommendation tasks and have achieved very appealing performance. However, most GNN-based recommendation methods suffer from the problem of data sparsity in practice. Meanwhile, pre-training techniques have achieved great success in mitigating data sparsity in various domains such as natural language processing (NLP) and computer vision (CV). Thus, graph pre-training has great potential to alleviate data sparsity in GNN-based recommendations. However, pre-training GNNs for recommendations face unique challenges. For example, user-item interaction graphs in different recommendation tasks have distinct sets of users and items, and they often present different properties. Therefore, the successful mechanisms commonly used in NLP and CV to transfer knowledge from pre-training tasks to downstream tasks such as sharing learned embeddings or feature extractors are not directly applicable to existing GNN-based recommendations models. In this chapter, we aim to effectively pre-training GNNs for the recommendations task. To tackle the aforementioned challenges, we delicately design an adaptive graph pre-training framework (ADAPT) for the localized collaborative filtering method proposed in Chapter 4. It does not require transferring user/item embeddings and is able to capture both the common knowledge across different graphs and the uniqueness of each graph simultaneously. Extensive experimental results have demonstrated the effectiveness and superiority of ADAPT.

5.1 Introduction

Recommendation is one of the most ubiquitous and successful applications of artificial intelligence in our daily life. It has been widely adopted in various online services such as target advertising and online shopping. Basically, recommendation systems aim to predict a user's preference based on his/her historical interactions with different items and further recommend to the user some items that he/she may have potential interest in [3]. Many existing solutions for recommendations follow the paradigm of first learning a set of latent factors (i.e., embeddings for users and items) and then building an interaction function to make recommendation decisions based on the learned embeddings. Matrix factorization (MF) is a representative method of such techniques. It aims to learn user and item embeddings directly from the user-item interaction matrix and then make predictions via inner product over the user and item embeddings. In recent years, we have witnessed increasing efforts incorporating deep neural networks to advance these techniques via refining the user and item embeddings [36] and modeling interaction functions [108, 25].

The interactions among users and items in a recommendation task can be naturally denoted as a user-item bipartite graph. From this perspective, the key for a recommendation task is to learn the node representations from the user-item bipartite graph. Graph neural networks (GNNs), which generalize deep neural networks (DNNs) to graph data, have been theoretically and empirically proved to be very powerful in representation learning for graph data [119, 135]. Therefore, there is increasing attention on adopting GNNs in addressing recommendation tasks. GNNs are able to inherently capture important high-order user-item connectivity in a given user-item interaction bipartite graph, as a consequence, they can boost the recommendation performance. For example, NGCF [110] proposed to propagate embeddings over user-item graphs based on the message-passing framework of GNNs and achieved significant performance improvement. LightGCN [35] further refined the design of GNNs for recommendation tasks by eliminating the feature transformation and non-linear activation and achieved state-of-the-art performance. Despite achieving great success in recommendation tasks, we empirically found that current GNN-based recommendation methods suffer from a common and practical problem: data scarcity[1]. In other words, when the user-item interaction graphs are sparse, the performance of most existing GNN-based recommendations tends to drop substantially. Unfortunately, historical user-item interactions in the real-world recommendations are often scarce [3]. The challenge of data scarce is also universal in other domains such as NLP [85] and CV [32, 44], where pre-training techniques have been proposed to alleviate this problem. Typically, a model is first pre-trained on a large dataset with abundant label information (either self-supervised label or supervised label), and then finetuned over the

[1]More details about how the data sparsity affects the performance of existing GNN-based recommendation methods can be found at Section 5.3.

downstream datasets with limited label information. Pre-training has been demonstrated to be effective in alleviating the data scarcity issue of the downstream task by transferring knowledge from the pre-training data. Thus, it is natural to ask: *can we also leverage pre-training techniques to facilitate GNN-based recommendation models?*

Adopting pre-training for existing GNN-based recommendations faces unique challenges. The great success of pre-training in NLP and CV relies on the effective mechanisms that enable knowledge sharing or transferring from the pre-trained tasks to the downstream tasks. In NLP, the vocabulary of words or tokens is shared; therefore, both context-independent embeddings (e.g., word2vec [72]) and context-sensitive embeddings (e.g., GPT [86] and BERT [15] from pre-training tasks can be transferred. In CV, low-/mid-level features (such as edges, and textures) given by the pre-trained model can be leveraged by the fine-tuned tasks. However, user and item embeddings denote the major parameters of existing GNN-based recommendation models. Moreover, in recommendations, different interaction graphs often do not have the same sets of users and items. Therefore, these uniquenesses determine that the effective mechanisms from NLP and CV do not apply to existing GNN-based recommendations. In addition, a large amount of data is essential for the effectiveness of pre-training [132] in NLP and CV. For instance, the state-of-the-art pre-trained model in CV is typically pre-trained on tens of millions of images [14], and the pre-training dataset for NLP often consists of more than 1000M words [15]. Therefore, it is desirable to take advantage of many interaction graphs for pre-training. However, achieving this goal is challenging given that different graphs have distinct properties such as the number of items (or users) and graph density. As a consequence, dedicated efforts are required to tackle these unique challenges.

In this work, we propose an **Ada**ptive graph **P**re-training framework for localized collaborative fil**T**ering (**ADAPT**) [113] that provides effective solutions to address the aforementioned challenges. It consists of two key components: a meta-localized GNN (or meta-LGNN) model and the GNN adaptor. The meta-LGNN model provides a new perspective to build GNN-based recommendations, where it is trained to make recommendation predictions based on the neighboring structures of the target user and item, instead of learning a set of user and item embeddings and an interaction function. The rationality of this design is: the key collaborative filtering information of a given target user and target item can be encoded by their neighboring structure, which consists of their historical interactions. With meta-LGNN, there is no need to transfer user or item embeddings across different graphs.

To leverage multiple pre-training graphs with distinct properties, we design the GNN adaptor as a strategy to capture their differences. Specifically, given a recommendation task, the GNN adaptor can adapt the meta-LGNN model to a customized GNN model by considering the properties of its corresponding interaction graph. We conduct extensive experiments on various datasets, and the empirical results demonstrate the effectiveness and superiority of the proposed framework.

The remainder of the paper is structured as follows. In Section 5.2, we describe the proposed framework in detail, including the problem definition, framework overview, the basic model, and two key processes. We introduce the experiments to validate the effectiveness in Section 5.3, including experimental settings, preliminary study, performance comparison, ablation study and further probing. In Section 5.4, we then review some important related works. We conclude the paper with discussions on future works in Section 5.5.

5.2 The Proposed Framework

In this section, we first formally define the GNN pre-training problem for GNN-based recommendations. Then we describe an overview of the proposed ADAPT. Next, we introduce the GNN-based recommendation method for localized collaborative filtering as the basis of the proposed framework. Finally, we detail the pre-training process and the fine-tuning process.

5.2.1 The GNN Pre-Training Problem on Recommendations

We first briefly describe the problem studied in this paper. It is common that user-item interaction graphs are sparse in the real-world recommendations [3]. It is undoubtedly very challenging to perform GNN-based recommendations on a sparse user-item interaction graph. Meanwhile, there exist a large amount of user-item bipartite graphs from other recommendation tasks, which contain abundant information and knowledge. A natural idea to mitigate the data sparsity problem is to transfer useful information from existing graphs to facilitate the recommendation task on the target sparse graph. Recent years have witnessed the great success of the pre-training techniques to transfer knowledge from a large amount of data to help solve the target task in NLP and CV [86, 15]. Therefore, in this paper, we mainly focus on leveraging the pre-training strategy to address the data sparsity challenge in GNN-based recommendations.

Next, we formally define the pre-training problem for GNN-based recommendations. We denote the user-item interaction data in a recommendation task as a user-item bipartite graph $G = (U, I, E)$, where $U = \{u_1, u_2, \cdots, u_{|U|}\}$ represents the user set, $I = \{i_1, i_2, \cdots, i_{|I|}\}$ is the item set, and $E \subset U \times I$ indicates interactions between users and items. Each element e_{jk} in E suggests that there exists an interaction between the user u_j and the item i_k. We assume that there are N graphs for pre-training that are denoted $\{G_1, G_2, \cdots, G_N\}$. We further use $G_t = (U_t, I_t, E_t)$ to denote the target graph for the downstream recommendation task.

Generally, a GNN model consists of L layers, where the key component in each layer l is a graph convolution with parameter $\boldsymbol{\theta}^l_{conv}$. Thus, we can denote the parameters for a GNN model as $\boldsymbol{\Theta}_{GNN} = \{\boldsymbol{\theta}^1_{conv}, \cdots, \boldsymbol{\theta}^L_{conv}\}$. With above definitions, the goal of the pre-training task is to train a model from graphs $\{G_1, G_2, \cdots, G_N\}$ that can help build a GNN model $\mathcal{G}(\cdot; \boldsymbol{\Theta}_{GNN_t})$ for G_t from the downstream recommendation task.

5.2.2 An Overall Design of ADAPT

An overview of ADAPT is demonstrated in the top subgraph of Figure 5.1. It consists of two key components – a meta-LGNN model and a GNN adaptor. One key challenge of pre-training GNNs for recommendations is that user-item interaction graphs in different recommendation tasks have distinct user/item sets; as a result, it is hard to directly transfer user/item embeddings across different recommendation graphs like pre-trained word embeddings in NLP. To solve this challenge, we propose to use the meta-LGNN model based on a new collaborative filtering method [112]. It provides a new perspective for GNN-based recommendations, where the recommendation prediction is made based on the local structure surrounding a user and an item, rather than their embeddings, via a GNN model. Therefore, it does not need to learn embeddings of users and items, which is

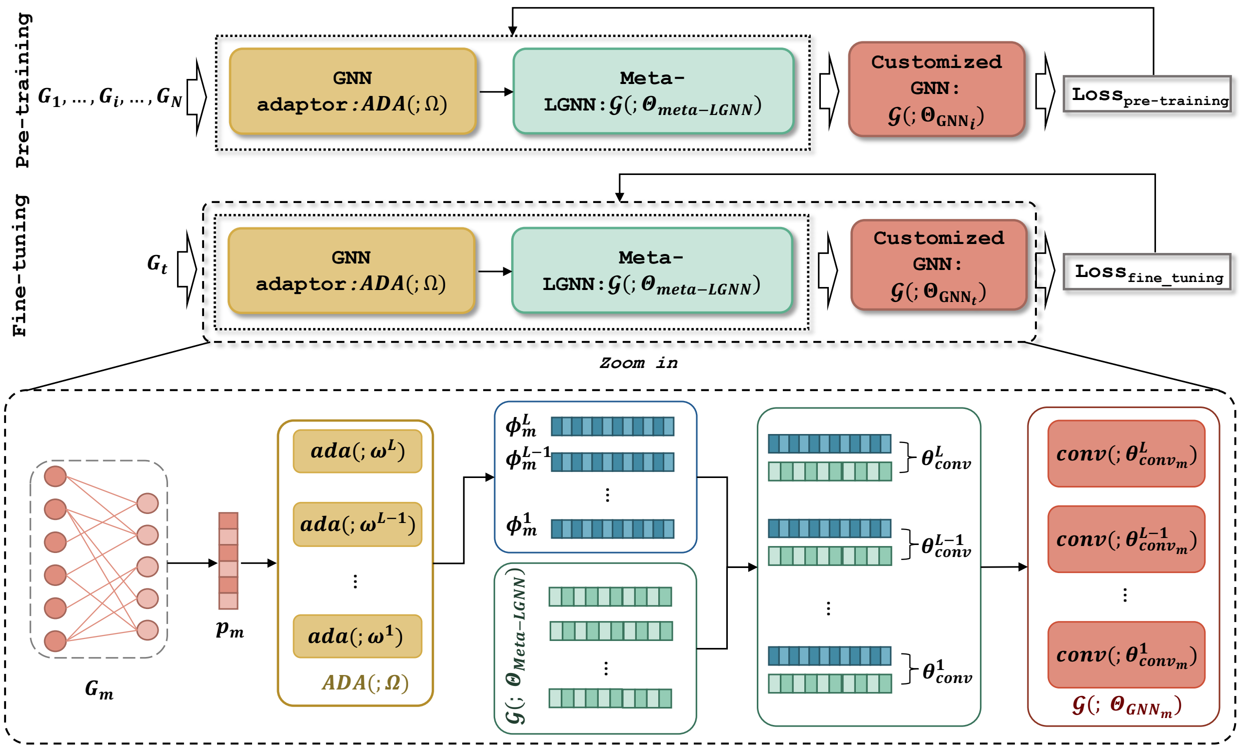

Pre-training
$G_1, \ldots, G_i, \ldots, G_N$
GNN adaptor : $ADA(;\Omega)$
Meta-LGNN : $\mathcal{G}(;\Theta_{meta-LGNN})$
Customized GNN : $\mathcal{G}(;\Theta_{GNN_i})$
$\text{Loss}_{\text{pre-training}}$
Fine-tuning
G_t
GNN adaptor : $ADA(;\Omega)$
Meta-LGNN : $\mathcal{G}(;\Theta_{meta-LGNN})$
Customized GNN : $\mathcal{G}(;\Theta_{GNN_t})$
$\text{Loss}_{\text{fine\_tuning}}$
Zoom in
G_m
p_m
$ada(;\omega^L)$
$ada(;\omega^{L-1})$
$ada(;\omega^1)$
$ADA(;\Omega)$
ϕ_m^L
ϕ_m^{L-1}
ϕ_m^1
$\mathcal{G}(;\Theta_{Meta-LGNN})$
θ_{conv}^L
θ_{conv}^{L-1}
θ_{conv}^1
$conv(;\theta_{conv_m}^L)$
$conv(;\theta_{conv_m}^{L-1})$
$conv(;\theta_{conv_m}^1)$
$\mathcal{G}(;\Theta_{GNN_m})$

Figure 5.1: The top subgraph shows an overview of the proposed framework ADAPT. There are two key components in ADAPT – a meta-LGNN model and a GNN adaptor. The GNN adaptor is designed to adapt the meta-LGNN model to a customized GNN model for the interaction graph of a given recommendation task by considering its distinct properties. With these two components, ADAPT first trains a GNN adaptor and a meta-LGNN model simultaneously in the pre-training phase. Then, given the target graph from the downstream recommendation task in the fine-tuning phase, we generate a customized GNN model based on the properties of the target graph via the GNN adaptor and the meta-LGNN model. The bottom subgraph illustrates the adaptation process. Given a specific graph G_m, the GNN adaptor takes its graph property vector $\mathbf{p}_m$ as input, and outputs customized adapting parameters $\mathbf{\Phi}_m$ for G_m. Suppose that the meta-LGNN model consists of L GNN layers, we can denote its parameter as $\mathbf{\Theta}_{meta-LGNN} = \{\boldsymbol{\theta}_{conv}^1, \cdots, \boldsymbol{\theta}_{conv}^L\}$, where $\boldsymbol{\theta}_{conv}^l \in \mathbb{R}^{d^{l-1} \times d^l}$ and d^l denotes the dimension of node embeddings outputted by the l-th GNN layer. The GNN adaptor will generate L adapting parameters corresponding to these GNN layers for G_m, which can be denoted as $\mathbf{\Phi}_m = \{\boldsymbol{\phi}_m^1, \cdots, \boldsymbol{\phi}_m^L\}$, where $\boldsymbol{\phi}_m^l \in \mathbb{R}^{2d^{l-1}d^l}$. For the graph convolution in the l-th GNN layer of the customized GNN model for graph G_m, its parameters $\boldsymbol{\phi}_m^l$ are adapted via $\boldsymbol{\phi}_m^l$ on $\boldsymbol{\theta}_{conv}^l$. Note that the adaptation processes of the pre-training and fine-tuning processes are the same.

naturally more flexible for pre-training, compared to most existing GNN-based recommendation methods. The success behind pre-training is that there exists common knowledge that can be transferred from the pre-training data to the downstream task. On the one hand, there are similar patterns of user-item local structures in different recommendation graphs, which are captured by the proposed meta-LGNN model. However, there could exist different patterns for these graphs since different graphs can present distinct properties. Therefore, ADAPT provides the GNN adaptor that can adapt the meta-LGNN model to a customized GNN model for the interaction graph of a given recommendation task by considering its distinct properties. With these two components, ADAPT first trains a GNN adaptor and a meta-LGNN model simultaneously in the pre-training phase. Then, given the target graph from the downstream recommendation task in the fine-tuning phase, we generate a customized GNN model based on the properties of the target graph via the GNN adaptor and the meta-LGNN model. In the following subsections, we first introduce the proposed GNN recommendation method for localized collaborative filtering in ADAPT. Next, we describe the pre-training process and the fine-tuning process of ADAPT in detail.

5.2.3 GNN-based Recommendation Method for Localized Collaborative Filtering

Graph Neural Network (GNN) models have been demonstrated effective and superior in facilitating recommendation tasks [123, 110, 35]. Most existing GNN-based recommendation methods aim at learning a set of user and item embeddings via a GNN model. Given that user-item interaction graphs from different recommendation tasks have distinct user/item sets, these GNN-based recommendation methods are impractical to be pre-trained, since user/item embeddings can not be transferred across different graphs. To take advantage of the power of both GNN methods and the pre-training techniques, we propose to adopt a new GNN-based localized collaborative filtering, (i.e., meta-LGNN) to build the basic recommendation model in our pre-training framework, which does not require learning user/item embeddings and makes predictions based on local structures extracted from the interaction graphs. Next, we will first briefly introduce the most common existing GNN-based recommendation method, and then illustrate the meta-LGNN.

We use $\mathbf{H} = \{\mathbf{h}_{u_1}, \cdots, \mathbf{h}_{u_{|U|}}, \mathbf{h}_{i_1}, \cdots, \mathbf{h}_{i_{|I|}}\}$ to denote the user and item embeddings the model aims to learn. $\mathbf{\Theta}_{GNN} = \{\boldsymbol{\theta}^1_{conv}, \cdots, \boldsymbol{\theta}^L_{conv}\}$ represents the parameters to be learned for a L-layer GNN model, where $\boldsymbol{\theta}^l_{conv}$ denotes the parameters for the graph convolution in layer l. Generally, for a user u in a given user item bipartite graph, the graph convolution first aggregates item embeddings from its neighborhood, and then updates the target user embedding based on its original embeddings

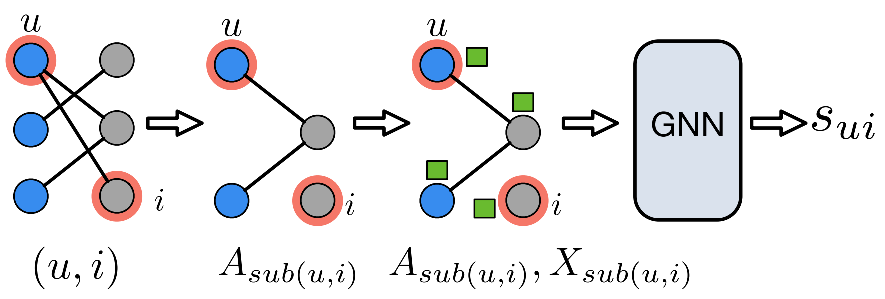


Figure 5.2: The proposed GNN recommendation method based on localized collaborative filtering. Given a target user u and a target item i , we first extract a local graph at these two target nodes from the user-item interaction graph, of which the structure is denoted as $\mathbf{A}_{sub(u,i)}$. Next, we generate a positional attribute for each node based on their minimum distances towards the target nodes, and these node attributes are denoted as $\mathbf{X}_{sub(u,i)}$. Then, we can leverage a GNN model to get the graph representation for this attributed graph. Finally, we can compute a recommendation score s_{ui} based on the local graph representation via a scoring function to make the final prediction for the given user and item pair.

and the aggregated embeddings. The update process of item embeddings works similarly as that for a user node. We formalize the user node embedding update process via the graph convolution in the l-th layer of GNN for illustration as follows:

$$\mathbf{h}_u^l = f_{conv}(\mathbf{h}_p^{l-1}, \mathbf{h}_u^{l-1}; \theta_{conv}^l), \forall p \in \mathcal{N}(u), \tag{5.1}$$

where f_{conv} represents the graph convolution operation and $\forall p \in \mathcal{N}(u)$ denotes any item p belonging to the one-hop neighborhood of user u. Note that $\mathbf{h}_u^0 = \mathbf{h}_u$ and $\mathbf{h}_i^0 = \mathbf{h}_i$. The refined user embedding $\mathbf{h}_u^L$ and the item embedding $\mathbf{h}_i^L$ outputted by the final layer of the GNN model are used to represent each user and item in a bipartite graph, and then typically we use the inner product over the user embedding $\mathbf{h}_u^L$ and item embedding $\mathbf{h}_i^L$ to make the recommendation prediction. Overall, these GNN-based recommendation methods aim at learning the user/item embeddings $\mathbf{H}$ and the GNN model parameters $\mathbf{\Theta}_{GNN}$ simultaneously. Since interaction graphs from different recommendation tasks have distinct sets of users and items, apparently it is not feasible to transfer the learned embeddings, i.e., $\mathbf{H}$.

In this paper, in order to ease pre-training for GNN-based recommendations, we build the meta-LGNN model based on a new collaborative filtering method [112]. The intuition is that the key collaborative filtering information for recommendation is encoded in the historical interactions.

Given a specific user and item pair, their key collaborative information is also encoded in their historical interactions. Thus it is reasonable to make a recommendation prediction based on the local structure consisting of their historical interactions. The framework of meta-LGNN is shown in Figure 5.2. For a target user and item pair, we first extract a local graph at these two target nodes from the user-item interaction graph. Next, we generate a positional attribute for each node based on their minimum distances toward the target nodes. Then, we can leverage a GNN model to get the graph representation for this attributed graph. Finally, we can compute a recommendation score based on the local graph representation via a scoring function to make the final prediction for the given user and item pair.

Given a target user u and a target item i in a user-item interaction graph G, we perform two random walks [80] starting from these two nodes on G, separately. Note that the random walk strategy we adopt in our work is with the restarting mechanism, and thus this process can be regarded as a neighbor sampling process, where we can get two sampled neighboring node sets $\mathcal{N}_u$ and $\mathcal{N}_i$ for u and i, respectively. Next, we merge these two node sets to get an overall node set $N_{(u,i)} = \mathcal{N}_u \cup \mathcal{N}_i$, and finally, we can get a local graph $G_{sub(u,i)}$ based on the set $N_{(u,i)}$ to reveal the local structure around the given pair (u, i). Note that the reason why we only sample some neighboring nodes is to ensure the scalability of our model on large graphs. Apart from graph structure $\mathbf{A}_{sub(u,i)}$, we also need its node attributes $\mathbf{X}_{sub(u,i)}$, so that we can use a GNN model $\mathcal{G}(\cdot; \mathbf{\Theta})$ to get a meaningful representation for $G_{sub(u,i)}$. For the model transferability, we propose to use the Double-Radius Node Labeling (DRNL) [129] to generate a set of positional attributes for each user or item in a graph, rather than using the existent node attributes, which are very likely to miss or be different across various recommendation tasks. The setting without no node attributes is also commonly adopted by previous related works [53, 110, 35], and it is straightforward for the proposed ADAPT framework to incorporate node attributes if they are available. DRNL labels each node based on its minimum distances towards the target user u and the target item i on the local graph $G_{sub(u,i)}$. Given a user/item node t, its positional attribute x_t can be calculated as:

$$x_t = 1 + \min(d_u, d_i) + (d/2)^2 \tag{5.2}$$

where d_u denotes the minimum distance between u and t, and d_i denotes the minimum distance between i and t. $d = (d_u + d_i)$ is the sum of two minimum distances. Note that we denote $x_u = 1$ and $x_i = 1$, so that we can distinguish the target user and item from other nodes. Now we have the graph structure and the node positional attributes of $G_{sub(u,i)}$, thus, we can use a GNN model $\mathcal{G}(\cdot; \mathbf{\Theta})$ to encode it into a meaningful representation. The node embedding update process in each GNN layer also follows Eq. 5.1. we use $\mathbf{H}^L_{sub(u,i)} \in \mathbb{R}^{n \times d}$ to denote the user-item embedding matrix outputted by the final GNN layer, where L is the number of GNN layers, n is the number of users and items in $G_{sub(u,i)}$ and d denotes the embedding dimension. Then, the GNN model leverages a pooling operation to get the graph representation for $G_{sub(u,i)}$ as follows:

$$\mathbf{h}_{(u,i)} = pool(\mathbf{A}_{sub(u,i)}, \mathbf{H}^L_{sub(u,i)}). \tag{5.3}$$

Note that the positional attributes are used as the input node features, i.e., $\mathbf{H}^0_{sub(u,i)} = \mathbf{X}_{sub(u,i)}$, which means that there is no need to learn the user/item embeddings. Finally, a score function is applied to compute a score for the user-item pair (u, i) to make a prediction. Overall, meta-LGNN aims at only learning the GNN model parameters $\mathbf{\Theta}_{GNN}$ while eliminating the user/item embeddings $\mathbf{H}$.

5.2.4 The ADAPT Pre-training

In this subsection, we will introduce the overall pre-training process of the proposed ADAPT. In this pre-training phase, one key challenge is how to learn the common transferable knowledge across multiple interaction graphs and capture their differences simultaneously. To solve this challenge, we equip ADAPT with a GNN adaptor. The basic idea of model adaptation is to adapt the meta-LGNN model to a customized GNN model specially for each recommendation graph based on its graph properties. In other words, we aim at customizing similar GNN models for recommendation graphs sharing similar graph properties. One key factor for the rationality of the designed model is that the graph properties of a recommendation graph are related with the collaborative filtering patterns. Thus, it is important to choose appropriate graph properties that can indicate collaborative filtering information. Intuitively, recommendation graphs holding similar structure properties are more likely to have analogical recommendation patterns. For examples, large online-shopping platforms may share more similar collaborative filtering patterns with each other than that with small online-shopping platforms. Recommendation domains where the purchasing behaviors are frequent may have different recommendation patterns with that where the users are inactive. More details about the design of graph properties can be found in Section 5.2.4.1.

In the pre-training process, both the GNN adaptor and the meta-LGNN model are optimized simultaneously. Next, we will detail the GNN adaptor component, the adaption process and the pre-training process.

5.2.4.1 The GNN Adaptor

The goal of the GNN adaptor is to generate customized adapting parameters to adapt the meta-LGNN model for a given graph. Given a specific graph G_m, the GNN adaptor takes its graph property vector $\mathbf{p}_m$ as input, and outputs customized adapting parameters $\mathbf{\Phi}_m$ for G_m. It is expected that the chosen graph properties are able to indicate important collaborative filtering information. To achieve this goal, we carefully select numerous graph structure properties. Intuitively, the collaborative filtering pattern is likely to be related to the recommendation graph size, i.e., the number of users and items in this recommendation graph. Also, the ratio between the number of users and that of items, and the frequency of the user-item interactions are important for recommendation patterns. Thus, we include the number of nodes, the number of edges, the user-item ratio, and the graph density in the graph properties. In addition, we also include the degree assortativity coefficient, which may indicate whether active users prefer popular items or niche items in a specific recommendation domain. Likewise, the robins-alexander clustering coefficient, the number of connected components, and the global efficiency have been included to indicate the interaction pattern of a recommendation graph. To conclude, to fully exploit the interaction pattern in a recommendation graph, we utilize eight normalized graph structural properties including the number of nodes, the number of edges, the user-item ratio, the graph density, the degree assortativity coefficient, robins-alexander clustering coefficient, the number of connected components and the global efficiency as the graph property input of the GNN adaptor. Note that there are also other alternatives we can explore to serve as the input of the GNN adaptor, such as the graph representation extracted by some graph embedding methods or unsupervised graph learning methods, which can also capture some important graph information. However, these alternatives may introduce additional model complexity and computing overhead. Thus, we prefer to use the eight graph properties introduced above, which are intuitive and empirically effective. As discussed before, suppose that the meta-LGNN model consists of L GNN layers, we can denote its parameter as $\mathbf{\Theta}_{meta-LGNN} = \{\boldsymbol{\theta}_{conv}^{1}, \cdots, \boldsymbol{\theta}_{conv}^{L}\}$, where $\boldsymbol{\theta}_{conv}^{l} \in \mathbb{R}^{d^{l-1} \times d^{l}}$ and d^{l} denotes the dimension of node embeddings outputted by l-th GNN layer. The GNN adaptor will generate L adapting parameters corresponding to these GNN layers for G_m, which can be denoted as $\mathbf{\Phi}_m = \{\boldsymbol{\phi}_m^{1}, \cdots, \boldsymbol{\phi}_m^{L}\}$, where $\boldsymbol{\phi}_m^{l} \in \mathbb{R}^{2d^{l-1}d^{l}}$. The GNN adaptor can be modeled using any function of $\mathbf{p}_m$. Specifically, for the graph convolution in the l-th GNN layer for G_m , the GNN adaptor generates its corresponding adapting parameters as follows:

$$\boldsymbol{\phi}_m^{l} = ada(\mathbf{p}_m; \boldsymbol{\omega}^{l}), \tag{5.4}$$

where $\boldsymbol{\omega}^{l}$ denotes the parameters of the GNN adaptor function $ada()$ corresponding to the graph convolution in the l-th GNN layer. In our work, we implement $ada()$ as a feed-forward neural network. Overall, we can summarize the adapting parameters generation process for G_m as follows:

$$\mathbf{\Phi}_m = ADA(\mathbf{p}_m; \mathbf{\Omega}), \tag{5.5}$$

where $ADA()$ consists of all adaptor functions for different GNN layers and $\mathbf{\Omega} = \{\boldsymbol{\omega}^{1}, \cdots, \boldsymbol{\omega}^{L}\}$ represents the parameters for all the GNN adaptors.

5.2.4.2 The Adaptation Process

In our work, the meta-LGNN model can be arbitrary GNN models including GCN [52], GIN [120] and etc [126, 69, 22]. The meta-LGNN model can be directly applied to a given user-item interaction graph.

However, as aforementioned, there exist both similarities and differences among the user-item interaction graphs in different recommendation tasks. Thus to preserve similarities and differences simultaneously, we do not directly apply the same meta-LGNN model to all the recommendation graphs in the pre-training process. Instead, we utilize the GNN adaptor to generate a customized GNN model for each recommendation graph based on the meta-LGNN model and its graph properties. The adaptation process is illustrated in the bottom subgraph of Figure 5.1. Specifically, for the graph convolution in the l-th GNN layer of the customized GNN model for graph G_m, its parameters are adapted as follows:

$$\boldsymbol{\theta}^l_{conv_m} = \boldsymbol{\theta}^l_{conv} \diamond \boldsymbol{\phi}^l_m, \tag{5.6}$$

where $\diamond$ denotes an adaptation operation. In the proposed model, we adopt FiLM [81] as the adaptation operation. Specifically, we split the adapting parameters $\boldsymbol{\phi}^l_m \in \mathbb{R}^{2d^{l-1}d^l}$ into two parts and reshape them as $\boldsymbol{\gamma}^l_m \in \mathbb{R}^{d^{l-1}\times d^l}$ and $\boldsymbol{\beta}^l_m \in \mathbb{R}^{d^{l-1}\times d^l}$. Then, we can formalize the adapting process of $\boldsymbol{\theta}^l_{conv}$ for G_m as follows:

$$\boldsymbol{\theta}^l_{conv_m} = \boldsymbol{\theta}^l_{conv} \odot \boldsymbol{\gamma}^l_m + \boldsymbol{\beta}^l_m, \tag{5.7}$$

where $\odot$ denotes the element-wise multiplication between two matrices. By applying the adaptation operation on L GNN layers of the meta-LGNN model, we can get the customized GNN model $\mathcal{G}(\cdot; \boldsymbol{\Theta}_{GNN_m})$ for G_m, where $\boldsymbol{\Theta}_{GNN_m} = \{\boldsymbol{\theta}^1_{conv_m}, \cdots, \boldsymbol{\theta}^L_{conv_m}\}$. The overall adaptation for G_i can be summarised as:

$$\boldsymbol{\Theta}_{GNN_m} = \boldsymbol{\Theta}_{meta-LGNN} \Diamond \boldsymbol{\Phi}_m. \tag{5.8}$$

Then, we utilize the customized GNN model to generate graph embedding for the local graph $G_{sub(u,i)}$ extracted from G_m for any user-item pair (u, i) as follows:

$$\begin{aligned} \mathbf{h}_{u,i} = pool(\mathbf{A}_{sub_{u,i}}, \mathcal{G}(\mathbf{A}_{sub_{u,i}}, \mathbf{X}_{sub_{u,i}}; \boldsymbol{\Theta}_{GNN_m})), \\ \forall (u, i) \in E_m \end{aligned} \tag{5.9}$$

5.2.4.3 The Pre-training Process

In the pre-training phase, suppose that there are N user-item interaction graphs $\{G_1, G_2, \cdots, G_N\}$ from different recommendation tasks. For any graph G_m, we can generate a customized GNN model $\mathcal{G}(; \boldsymbol{\Theta}_{GNN_m})$ based on the meta-LGNN model and graph properties $\mathbf{p}_m$ via the GNN adaptor. For any user-item pair $(u, i) \in E_m$, we utilize $\mathcal{G}(; \mathbf{\Theta}_{GNN_m})$ to compute a graph representation $\mathbf{h}_{u,i}$ for its corresponding local graph $G_{sub_{u,i}}$, and then we use a scoring function to compute a recommendation score for (u, i) based on $\mathbf{h}_{u,i}$:

$$s_{(u,i)} = score(\mathbf{h}_{(u,i)}; \boldsymbol{\delta}) = \sigma(\mathbf{h}_{(u,i)} \cdot \boldsymbol{\delta}), \tag{5.10}$$

where $\boldsymbol{\delta}$ is the linear transformation weights to be learned in the scoring function. We adopt the pairwise BPR loss [90] in ADAPT. The BPR loss is one of the most popular objective functions in recommendation tasks, which measures the relative order of the positive node pairs and negative node pairs. The positive node pairs are user-item interactions observed in graphs, while the negative node pairs are non-existent user-item interactions generated by negative sampling. Specifically, BPR assumes that the recommendation score of the positive node pairs should be higher than the corresponding negative ones. The objective function $\mathcal{L}_{pre}$ of our model in the pre-training phase is formulated as follows:

$$\min_{\mathbf{\Omega}, \mathbf{\Theta}_{meta-LGNN}, \boldsymbol{\delta}} \sum_{(u,i,i^{'}) \in O} - \ln \sigma(s_{(u,i)} - s_{(u,i')}), \tag{5.11}$$

where $O = \bigcup_{m} \{(u, i, i^{'}) | (u, i) \in E_m, (u, i^{'}) \in E_m^-\}$ denotes the pre-training data for the graph set $\{G_1, G_2, \cdots, G_N\}$. E_m represents the existent interaction set between users and items in G_m and E_m^- is the non-existent interaction set. The overall pre-training process is summarized in Algorithm 1. Given a set of interaction graphs used for pre-training $\{G_1, G_2, \cdots, G_N\}$, batch size b and the amount of interactions sampled for pre-training $N_{samples}$, we first randomly initialize the meta-LGNN model, the GNN adaptor and the scoring function. Next, in each training epoch (from line 3 to line 12), a random graph G_m is sampled from $\{G_1, G_2, \cdots, G_N\}$ and the GNN adaptor generates its corresponding adaptation parameters $\mathbf{\Phi}_m$ based on its structure properties $\mathbf{p}_m$, and then the customized GNN model $\mathcal{G}(; \mathbf{\Theta}_{GNN_m})$ is generated especially for G_m. Furthermore, we randomly sample b user-item interactions from the pre-training graph G_m and generate their negative counterparts, and then the meta-LGNN model, the GNN adaptor and the scoring function are updated via minimizing the BPR loss on these samples. The training process will continue until the objective function is converged.

Algorithm 1 The Pre-training Process of ADAPT

```
Input:
    A set of interaction graphs from different recommendation tasks: $\{G_1, G_2, \cdots, G_N\}$;
    Batch size $b$;
    Sample amount $N_{samples}$ (Note that $N_{samples} < \sum_m |E_m|$.)
Output:
    The meta-LGNN model $\mathcal{G}(;\mathbf{\Theta}_{meta-LGNN})$;
    the GNN adaptor $ADA(;\mathbf{\Omega})$;
    The scoring function $score(;\boldsymbol{\delta})$
 1: Initialize $\mathbf{\Theta}_{meta-LGNN}$, $\mathbf{\Omega}$ and $\boldsymbol{\delta}$ randomly;
 2: while not converged do
 3:     for batch=1,..,$\frac{N_{samples}}{b}$ do
 4:         Sample a graph $G_m$ from $\{G_1, G_2, \cdots, G_N\}$;
 5:         Compute $\mathbf{\Phi}_m = ADA(\mathbf{p}_m; \mathbf{\Omega})$;
 6:         Compute $\mathbf{\Theta}_{GNN_m} = \mathbf{\Theta}_{meta-LGNN} \Diamond \mathbf{\Phi}_m$;
 7:         Sample $b$ user-item interactions $(u, i) \in E_m$;
 8:         Sample $b$ negative interactions $(u, i') \notin E_m$;
 9:         Compute $loss = \sum -\ln \sigma(score_{(u,i)} - score_{(u,i')})$
10:         Compute $grad = backward(loss)$
11:         Update( $\mathbf{\Theta}_{meta-LGNN}, \mathbf{\Omega}, \boldsymbol{\delta}$; $grad$)
12:     end for
13: end while
```

5.2.5 The ADAPT Fine-tuning

In the fine-tuning phase, we also adopt the BPR loss. However, in this phase, we aim at optimizing the parameters of the customized GNN model. The formulation is listed below:

$$\min_{\mathbf{\Theta}_{GNN_t}} \sum_{(u,i,i')\in E_t^{'}} -\ln \sigma(\hat{y}_{(u,i)} - \hat{y}_{(u,i')}), \tag{5.12}$$

where $E_t^{'} = \{(u, i, i^{'})|(u, i) \in E_t, (u, i^{'}) \in E_t^{-}\}$ denotes the training data for the target graph G_t. E_t represents the set of existent interactions between users and items in G_t and E_t^{-} is the non-existent interaction set generated manually.

With the GNN adaptor and the meta-LGNN model from the pre-training phase, we can conduct model fine-tuning especially for a target recommendation graph G_t. We design two fine-tuning strategies and illustrate them in Figure 5.3.

- The first fine-tuning strategy is named as *direct fine-tuning*. As shown in the top subfigure of

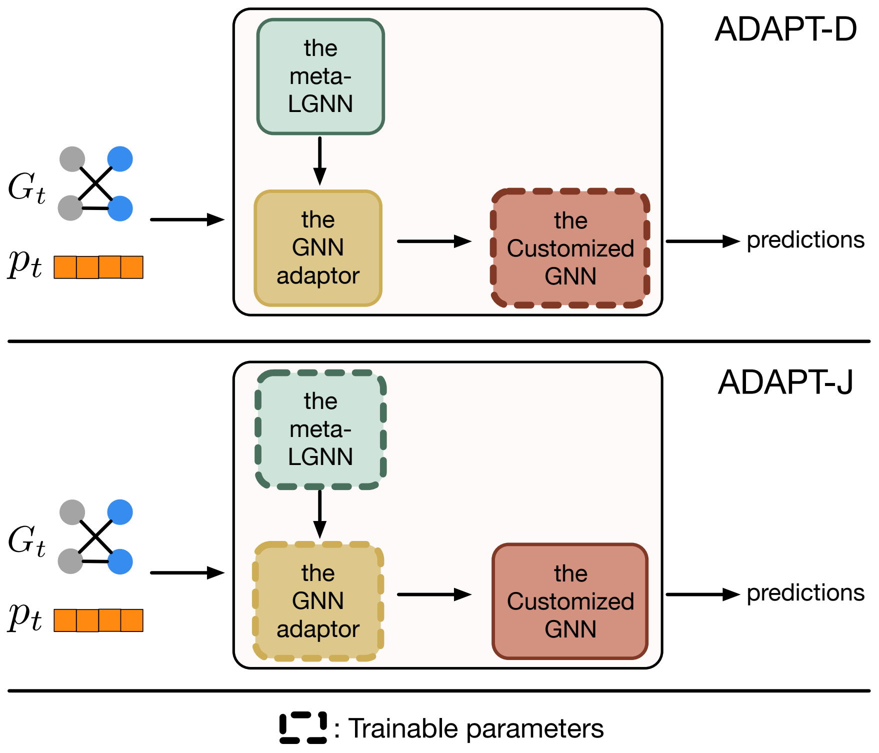


Figure 5.3: The fine-tuning strategies. We design two fine-tuning strategies – *direct fine-tuning* and *joint fine-tuning*, which are denoted as *ADAPT-D* and *ADAPT-J*, separately. In *ADAPT-D*, we generate the customized GNN model $\mathbf{\Theta}_{GNN_t}$ based on the input graph via the GNN adaptor and the meta-LGNN, and we fine-tune the customized GNN model $\mathbf{\Theta}_{GNN_t}$; In *ADAPT-J*, we fine-tune both the meta-LGNN model $\mathbf{\Theta}_{meta-LGNN}$ and the GNN adaptor $\mathbf{\Omega}$.

Figure 5.3, given a target downstream graph G_t, we first compute the customized adapting parameters $\mathbf{\Phi}_t = ADA(p_t; \mathbf{\Omega})$ based on its property vector p_t via the GNN adaptor $ADA(; \mathbf{\Omega})$, then generate the customized GNN model $\mathbf{\Theta}_{GNN_t} = \mathbf{\Theta}_{meta-LGNN} \Diamond \mathbf{\Phi}_t$. Finally, we fine-tune $\mathbf{\Theta}_{GNN_t}$ by optimizing the objective function described in Eq. 5.12. We denote ADAPT with this fine-tuning strategy as *ADAPT-D*.

- The second fine-tuning strategy is named as *joint fine-tuning*. As shown in the bottom subfigure of Figure 5.3, given a target downstream graph G_t, we fine-tune both the meta-LGNN model $\mathbf{\Theta}_{meta-LGNN}$ and the GNN adaptor $\mathbf{\Omega}$ on G_t. This fine-tuning strategy is inspired by test-time training [100], which is proposed to fine-tune the model for a test sample in the test process so that the model can be better adapted for the test data. We use *ADAPT-J* to indicate ADAPT with the joint fine-tuning strategy.

5.2.6 Time Complexity Analysis

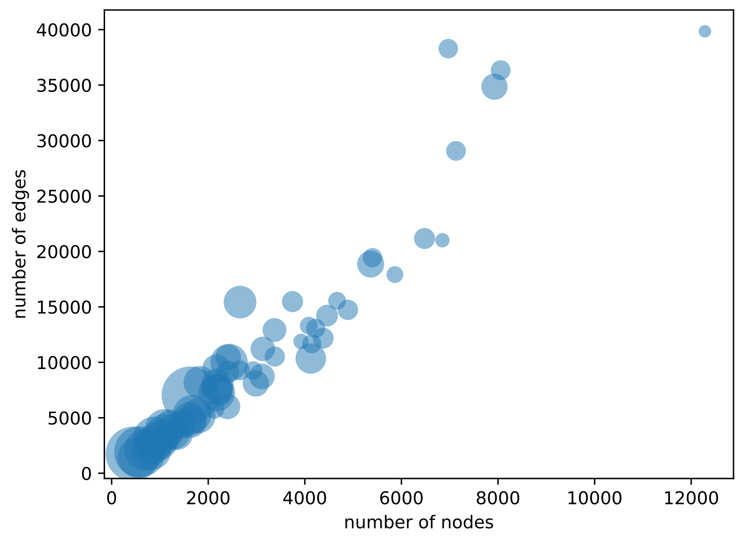
In the subsection, we analyze the additional time complexity introduced by the GNN adaptor in the ADAPT framework. For convenience, we assume the dimension of the output features from each meta-LGNN layer as d. Correspondingly, the dimension of output features from each GNN adaptor layer is $2d$. The graph property vector $\mathbf{p}_m$ of a given specific graph G_m is assumed to be p-dimensional. Then, the time cost of generating the adaptation parameters in each ADAPT layer is $O(d \cdot p)$ and the time complexity of the adaptation process is $O(d^2)$. Thus, for the ADAPT framework consisting of L layers, the overall time complexity caused by adaptation is $O(L \cdot p \cdot d + L \cdot d^2)$. The time complexity of the filtering operation in each layer of the meta-LGNN model is $O(N_e \cdot d + N_n \cdot d^2)$, where N_e and N_n denote the number of edges and the number of nodes of the input graph, respectively. Hence, the overall time complexity of the meta-LGNN model is $O(L \cdot N_e \cdot d + L \cdot N_n \cdot d^2)$. Typically, p is significantly smaller than N_e. Therefore, the additional time complexity introduced by the GNN adaptor in the ADAPT framework is comparatively low.

5.3 Experiments

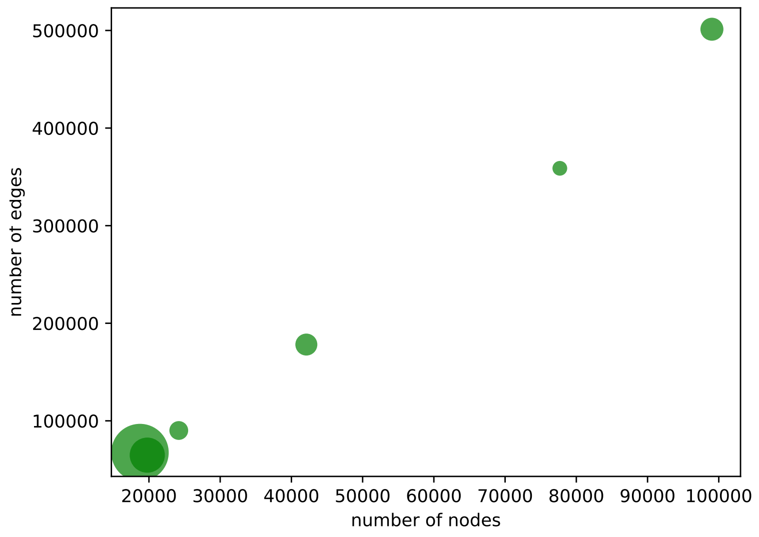
In this section, we conduct extensive experiments to validate the effectiveness of the proposed ADAPT. We first introduce the experimental settings. Then we illustrate how the sparsity of the recommendation graphs affects the performance of existent GNN-based recommendation methods. Next, we evaluate the performance of ADAPT and representative baselines on various real-world datasets. We conduct the ablation study to understand the importance of the GNN adaptor. Finally, we investigate the impact of graph sparsity and the number of pre-training graphs on the performance of ADAPT.

5.3.1 Experimental Settings

In this work, we conduct experiments on datasets from two real-world applications: Tianchi [2] and MovieLens [1]. Specifically, we construct numerous user-item interaction graphs from different recommendation scenarios based on the item category from these two applications. For each application, we select some interaction graphs as pre-training graphs and some of them as the target



(a) 60 pre-training graphs from Tianchi.



(b) 6 pre-training graphs from MovieLens.

Figure 5.4: Properties of pre-training graphs. Note that each circle represents a graph and its size denotes the density of the graph.

downstream graphs. The statistics of the pre-training graphs are demonstrated in Figure 5.4 and these of the downstream graphs are summarized in Table 5.1. As shown in Figure 5.4, we select 60 interaction graphs from Tianchi and 6 interaction graphs from MovieLens for pre-training, and these graphs show very diverse properties. We have also summarized the graph property statistics of the

Table 5.1: Statistics of the downstream graphs from Tianchi and MovieLens. Note that we do not include all the graph properties in this table, since some properties vary with the graph data split method.

Dataset	#Users	#Items	#user-item ratio	#Edges
Tianchi-174490	2,267	285	7.954385965	3,826
Tianchi-61626	2,305	764	3.017015707	3,270
Tianchi-3937919	1,846	158	11.683544304	2,980
Tianchi-2798696	2,579	94	27.436170213	2,988
MovieLens-War	2,970	89	33.370786517	4,240

60 pre-training graphs from Tianchi in Appendix D.1. We briefly introduce the Tianchi dataset and the MovieLens dataset as follows:

- **Tianchi**: It is a user behavior dataset from *Alibaba* (one of the biggest online shopping platforms in China), which consists of millions of user-item interactions, such as clicking, liking, and purchasing. Each interaction record includes user ID, item ID, behavior type, item category ID, and interaction timestamp. In our work, we use Tianchi-ID to denote the user-item interaction graph whose items belong to the category ID.

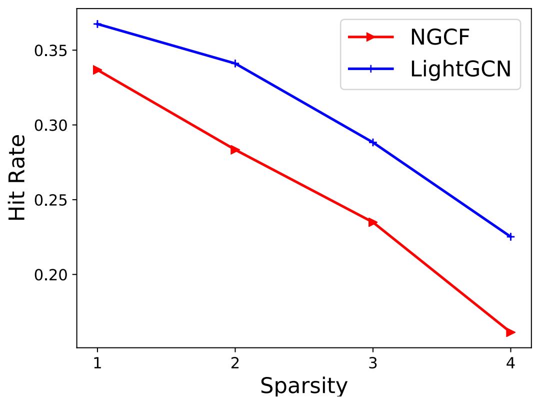
- **MovieLens**: This is a movie rating dataset from *MovieLens* (a popular movie recommendation website), which consists of user rating records for thousands of movies from different categories. Each rating record includes user ID, movie ID, rating, and movie category. In our work, we use MovieLens-X to denote the user-movie interaction graph whose movies come from the category X.

We mainly compare the proposed method with baseline methods from two groups. The first group includes GNN-based recommendation methods and a classic CF method. Particularly, we select NGCF and LightGCN since they are two of the most representative GNN-based recommendation methods, and LightGCN is one of the state-of-the-art models. MF is chosen because it is one of the most classic and popular recommendation methods. The second group includes existing pre-training methods for GNNs. Though there are a few pre-training methods for GNNs [40, 41, 84], the majority of them are not designed specifically to recommendations. Thus, we adapt a recent method GCC for recommendations as the representative baseline since it only relies on the topological information and can be pre-trained on multiple graphs as the proposed ADAPT does. We have not chosen the pre-training strategies proposed in [40] because there are no node attributes or graph labels in our scenarios. Similarly, GPT-GNN [41] has not been included because it also requires node attributes and cannot be applied across multiple graphs. The details of these baselines are presented in Appendix D.2. Note that since we do not focus on the cold-start problem, we do not include the pre-training work in [29] as one baseline.

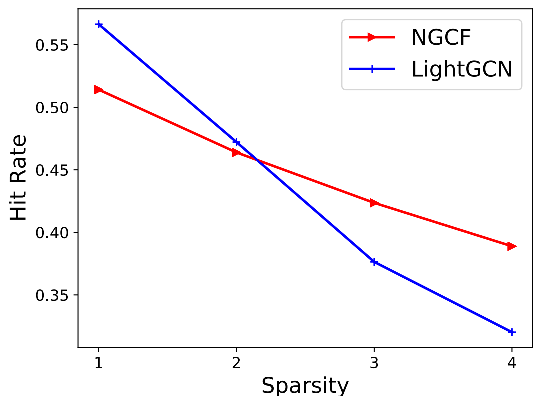
For the downstream recommendation tasks, we divide the corresponding user-item interactions into three sets: the training set, the validation set, and the test set. Note that each of the validation sets and the test set has 5 percent of the total samples. To avoid the cold-start problem, we constrain that all the users and items in the validation set and the test set should exist in the training set. In the fine-tuning phase, we use the validation set to select the best model and then report its performance in the test set. For each user-item interaction in the validation or the test set, we generate 49 non-existent user-item interactions for the user. In the model evaluation, we first compute a recommendation score for each interaction, and then we rank these scores. We calculate the Hit Rate (HR) of the model prediction based on if the score of the real user-item interaction is in the top 5 among all the 50 user-item interactions. For each experiment, we average and report the model performance with 5 seeds in terms of HR. More implementation details can be found in Appendix D.3

5.3.2 Preliminary Study

In this subsection, we study how the performance of two representative GNN-based recommendation methods, i.e., NGCF and LightGCN, is affected by the sparsity of interaction graphs. Specifically, we increase the sparsity of interaction graphs by randomly removing parts of the training edges while fixing the test edges. The results are shown in Figure 5.5 where $x \in \{1, 2, 3, 4\}$ (i.e., the x-axis) is used to denote the graph sparsity and a larger x indicates a sparser graph. Both LightGCN and NGCF show a significant decreasing trend in terms of recommendation performance with the increase of graph sparsity. This empirically demonstrates that GNN-based recommendation methods suffer from data sparsity. This motivates us to take advantage of pre-training techniques to



(a) Tianchi-61626.



(b) Tianchi-174490.

Figure 5.5: The performance of NGCF and LightGCN vs. the graph sparsity on two Tianchi datasets (The reported performance is measured with HR in %).

alleviate this problem.

5.3.3 Performance Comparison

We compare the proposed ADAPT and baselines described aforementioned on five datasets, including four item categories from Tianchi and one movie category from MovieLens. The recommendation methods are divided into two groups: 1) the *training from scratch* group consists of MF, NGCF, and LightGCN, which are directly trained from scratch on the target downstream graphs; and 2) the *pre-training & fine-tuning* group consists of GCC and the proposed ADAPT with two fine-tuning strategies, i.e., ADAPT-D and ADPAT-J. To ease the comparison, we also show the best performance of methods from these two groups and compute the performance gain of the *pre-training & fine-tuning* group over the *training from scratch* group. Note that in order to simulate the data scarcity scenarios in real-world recommendation applications, after the validation set and the test set are determined, we randomly remove some interactions from the remaining interactions under the constraint of introducing no isolated users or items. Specifically, we tailor two training sets: One retains 60 percent of the remaining interactions, and the other one consists of 40 percent of interactions. The comparison results on these two scenarios are shown in Table 5.2 and Table 5.3, respectively. To compare the proposed ADAPT framework with more graph pre-training methods,

Table 5.2: The recommendation performance comparison with 60% as training (The reported performance is measured with HR in %).

Datasets	**Training from Scratch**				**Pre-training & Fine-tuning**				**Performance Gain**
	MF	NGCF	LightGCN	**Best**	GCC	ADAPT-D	ADAPT-J	**Best**	
Tianchi-174490	33.51±3.71	46.39±3.07	47.22±3.08	**47.22±3.08**	37.17±15.6	60.94±1.46	58.64±2.85	**60.94±1.46**	**+29.05%**
Tianchi-61626	17.30±3.14	28.34±5.52	34.11±3.52	**34.11±3.52**	33.37±3.4	50.06±3.18	49.81±3.16	**50.06±3.18**	**+46.77%**
Tianchi-3937919	21.75±3.95	26.71±4.59	40.2±3.31	**40.2±3.31**	35.57±12.12	57.72±2.31	59.6±1.97	**59.6±1.97**	**+48.25%**
Tianchi-2798696	30.81±5.22	35.97±2.71	28.99±3.86	**35.97±2.71**	45.1±3.4	47.38±2.15	46.84±0.28	**47.38±2.15**	**+31.72%**
MovieLens-War	62.94±4.25	63.41±3.06	62.94±4.07	**63.41±3.06**	64.93±5.13	73.93±1.18	75.99±0.72	**75.99±0.72**	**+19.84%**

Table 5.3: The recommendation performance comparison with 40% as training (The reported performance is measured with HR in %).

Datasets	**Training from Scratch**				**Pre-training & Fine-tuning**				**Performance Gain**
	MF	NGCF	LightGCN	**Best**	GCC	ADAPT-D	ADAPT-J	**Best**	
Tianchi-174490	30.73±4.35	42.36±2.96	37.64±3.24	**42.36±2.96**	33.40±1.36	57.70±2.49	60.07±3.2	**60.07±3.20**	**+41.82%**
Tianchi-61626	16.32±3.21	23.50±4.89	28.83±4.48	**28.83±4.48**	32.27±2.32	49.94±3.75	52.39±1.27	**52.39±1.27**	**+81.70%**
Tianchi-3937919	20.60±3.84	25.57±3.62	35.44±1.70	**35.44±1.70**	34.9±17.0	49.13±8.98	58.79±3.37	**58.79±3.37**	**+65.9%**
Tianchi-2798696	28.86±4.12	33.29±1.98	27.11±4.89	**33.29±1.98**	46.17±2.53	48.66±2.86	46.58±2.10	**48.66±2.86**	**+46.16%**
MovieLens-War	61.14±4.52	64.17±2.50	56.78±4.86	**64.17±2.50**	50.76±13.65	71.85±1.76	71.47±1.18	**71.85±1.76**	**+11.97%**

we have also pre-trained the Meta-LGNN models following the InfoGraph and MVGRL methods, and compare their performance on the Tianchi downstream datasets in Table 5.4. We can make the following observations:

- NGCF and LightGCN often perform better than MF. This observation is consistent with previous observations in [110, 35].
- The proposed ADAPT frameworks achieve great performance improvement over the recommendation models trained from scratch on the target downstream data. This demonstrates the effectiveness of ADAPT in alleviating the data scarcity problem in recommendation tasks.
- ADAPT achieves significantly better performance than GCC, InfoGraph, and MVGRL. Compared to these pre-training methods, ADAPT is designed specifically to the pre-training task for recommendations with the recommendation BPR objective and the adaptor to capture the differences among pre-training graphs. More investigations on the importance of the adaptor will be discussed in the following subsection.

Table 5.4: The recommendation performance comparison with graph pre-training methods on Tianchi datasets (The reported performance is measured with HR in %).

	Dataset	GCC	InfoGraph	MVGRL	ADAPT
40%	Tianchi-174490	33.40±1.36	33.3±19.45	47.8±13.7	60.07±3.20
	Tianchi-61626	32.27±2.32	34.97±10.2	22.45±4.44	52.39±1.27
	Tianchi-3937919	34.9±17.0	32.21+21.8	46.8±11.5	58.79±3.37
	Tianchi-2798696	46.17±2.53	31.95±8.28	43.4±1.55	48.66±2.86
60%	Tianchi-174490	37.17±15.6	32.56±22.5	53.7±18.1	60.94±1.46
	Tianchi-61626	33.37±3.4	31.90±11.9	35.54±6.84	50.06±3.18
	Tianchi-3937919	35.57±12.12	32.6±25.6	48.59±7.19	59.6±1.97
	Tianchi-2798696	45.1±3.4	31.1±12.36	47.0±2.14	47.38±2.15

Table 5.5: Ablation study of the GNN adaptor (The reported performance is measured with HR in %).

Datasets	**Pre-training & Fine-tuning**			**Training from Scratch**
	60 pre-training graphs		1 pre-training graph	
	ADAPT-best	ADAPT-w/o-adaptor	ADAPT-w/o-adaptor	ADAPT-w/o-adaptor-scratch
Tianchi-174490	60.07±3.2	32.25±9.43	43.04±14.56	53.92±10.58
Tianchi-61626	52.39±1.27	44.78±9.83	43.81±6.36	49.57±3.98
Tianchi-3937919	58.79±3.37	38.12±7.29	48.59±7.89	58.25±5.89
Tianchi-2798696	48.66±2.86	47.36±1.46	45.64±1.50	47.25±2.15
Average	54.98	40.63	45.27	52.24

- Both the *direct fine-tuning* strategy and the *joint fine-tuning* strategy are effective. The *joint fine-tuning* strategy is empirically shown to be more effective than the *direct fine-tuning* strategy in most cases. This observation could indicate that fine-tuning the adaptor in the test time has the potential to benefit the performance of the downstream recommendation task. This observation is consistent with that in [100].

- The performance gain of ADAPT from the best baseline performance is more significant under the 40% than 60%. This observation suggests that pre-training is a promising solution to tackle the data sparsity problem in recommendations.

5.3.4 Ablation Study

We conduct an ablation study to investigate the effectiveness of the GNN adaptor. The results are shown in Table 5.5. We use ADAPT-w/o-adaptor to denote the ADAPT framework without the GNN adaptor. We pre-train two model instances of ADAPT-w/o-adaptor with 60 pre-training graphs and 1 pre-training graph, respectively. We use ADAPT-w/o-adaptor-scratch to indicate that ADAPT-w/o-adaptor is trained from scratch with only the downstream graph. ADAPT-best denotes the ADAPT framework with any fine-tuning strategy that can achieve better performance. We can make the following observations:

- The overall performance of ADAPT-best is significantly better than that of the ADAPT-w/o-adaptor and ADAPT-w/o-adaptor-scratch. This validates (1) the effectiveness of the GNN adaptor for pre-training and (2) the importance of pre-training.
- The ADAPT-w/o-adaptor instance pre-trained on 60 pre-training graphs performs even worse than the instance pre-trained on a single pre-training graph. This shows that it is necessary to capture the differences among multiple pre-training graphs, and the GNN adaptor in the proposed ADAPT can capture these differences in the pre-training process.
- Both the ADAPT-w/o-adaptor instances cannot beat ADAPT-w/o-adaptor-scratch. This indicates that pre-training is not always helpful. It can even introduce performance degradation if not carefully designed.

To further demonstrate the importance of pre-training, we check if ADAPT can transfer knowledge from the pre-training graphs to the downstream recommendation task. To achieve this goal, we directly use the models generated from the pre-training without fine-tuning. In particular, we compare the performance of three different variants of the proposed GNN recommendation method for localized collaborative filtering. They include a randomly-initialized model, the meta-LGNN model from the pre-training phase, and the customized-GNN model generated by the meta-LGNN model and the GNN adaptor, especially for the target dataset. The results are demonstrated in Table 5.6. Overall, the meta-LGNN model performs better than the randomly-initialized model, and the performance of the customized-GNN model is significantly better than that of the other two variants. These observations demonstrate that the meta-LGNN model learns some useful knowledge from the pre-training process, and the GNN adaptor effectively adapts the meta-LGNN

Table 5.6: Performance comparison of three different variants of the proposed GNN recommendation method for localized collaborative filtering (The reported performance is measured with HR in %).

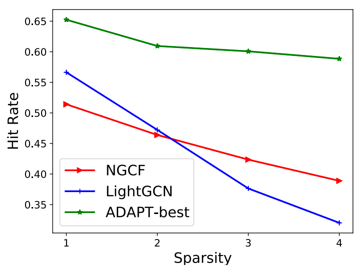
Datasets	random-ini	meta-LGNN	customized-GNN
Tianchi-174490	16.44	13.09	28.17
Tianchi-61626	12.64	16.20	26.99
Tianchi-3937919	16.11	12.89	30.33
Tianchi-2798696	11.68	19.33	17.59
Average	14.22	15.38	25.77

model to the customized one. The customized-GNN model performs remarkably better than the randomly-initialized model. This provides direct evidence that ADAPT successfully transfers knowledge from pre-training graphs to the downstream recommendation task.

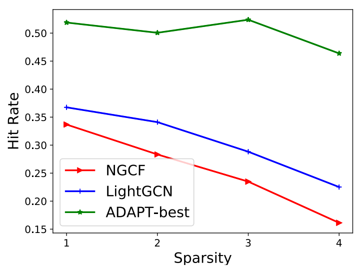
5.3.5 Further Probing

In this subsection, we further probe the proposed framework by exploring the following two problems: 1) how does the sparsity of the target downstream graph affect the ADAPT performance? and 2) how does the number of pre-training graphs influence the ADAPT performance?

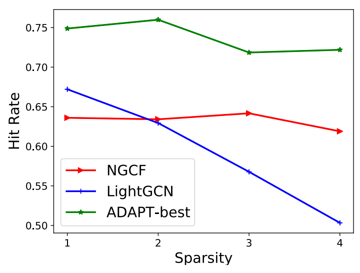
5.3.5.1 Sparsity Analysis



(a) Tianchi-174490.



(b) Tianchi-61626.

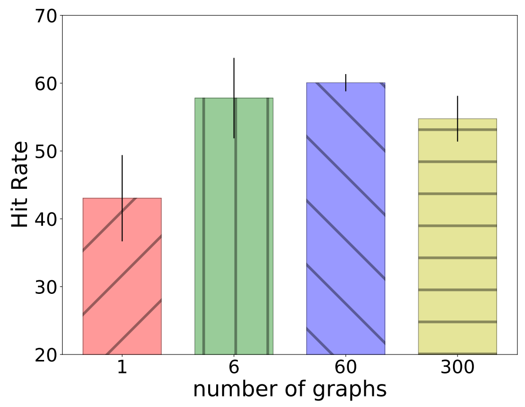


(c) MovieLens-War.

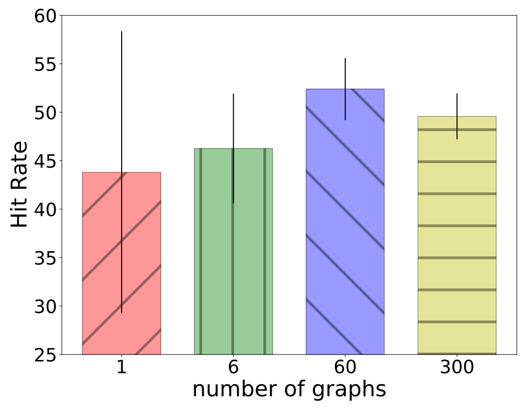
Figure 5.6: The performance of the proposed ADAPT with varied sparsity of the target downstream graph. Note that the x-axis is used to denote the graph sparsity. The larger the sparsity is, the sparser the target downstream graph is.

In Figure 5.6, we show how the performance of ADAPT and two GNN-based recommendation methods, NGCF and LightGCN, changes with the change of graph sparsity. Specifically, we gradually increase the graph sparsity by randomly removing some interactions from the training set under the constraint of introducing no isolated users or items, and meanwhile, the test set and validation set remain unchanged. It is observed that the proposed ADAPT performs much more stable with the increase of the graph sparsity, compared to NGCF and LightGCN. This observation further demonstrates the appealing of pre-training to mitigate the data sparsity problem in recommendations.

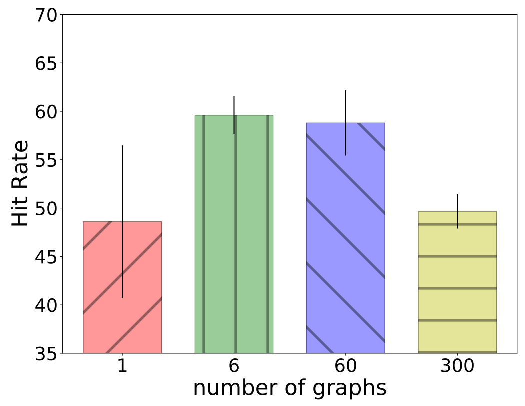
5.3.5.2 The Number of Graphs for Pre-training



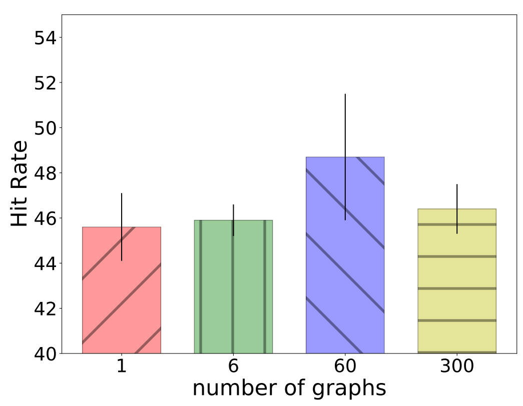
(a) Tianchi-174490.



(b) Tianchi-61626.



(c) Tianchi-393719.



(d) Tianchi-2798696.

Figure 5.7: Performance variants of the proposed ADAPT in terms of the number of pre-training graphs.

In order to explore the influence of the number of pre-training graphs on the model performance, we pre-train four ADAPT models on one single graph, 6 graphs, 60 graphs and 300 graphs, respectively. Then we fine-tune these models on four target downstream graphs. Note that for the

ADAPT model pre-trained on multiple graphs, we report its best performance among two fine-tuning strategies. For the ADAPT model pre-trained on one single graph, it is not applicable to pre-train the GNN adaptor, thus we report its performance without the GNN adaptor. As shown in Figure 5.7, the model performance first increases when the number of pre-training graphs increases and then it decreases consistently on all the datasets when the number of graphs is 300. This demonstrates that it is potentially beneficial to increase the number of pre-training graphs for the downstream performance. However, too many pre-training graphs may introduce noise that could hurt the downstream performance.

5.3.5.3 Investigation of Node Attributes Alternative

Following the commonly adopted scenarios [53, 110, 35] and also for the feasibility and transferability of the proposed ADAPT, we don't assume that there exists the same set of node attributes across different recommendation graphs. Instead, we propose to use Double-Radius Node Labeling (DRNL) [129] to generate a set of positional attributes, which are able to encode structural information, to serve as node attributes. The proposed ADAPT is very flexible with the input node attributes. If there do exist the same set of node attributes across different graphs, we can easily incorporate these raw node attributes by either concatenating them with the positional attributes or directly using them as input node attributes. To further investigate other alternatives for the input node attributes, we have also attempted to use the DeepWalk [82] embedding of each node as the input node attributes, and have compared its performance with the DRNL method on Tianchi dataset in Table 5.7. We can observe that the DeepWalk node attributes perform worse than the DRNL node attributes in most cases, but it beats its counterpart in one case. This shows that DRNL is a good attributes generation method for the proposed ADAPT given its effectiveness and simplicity, and there may exist potential to improve the model performance by further investigating other node attributes alternatives.

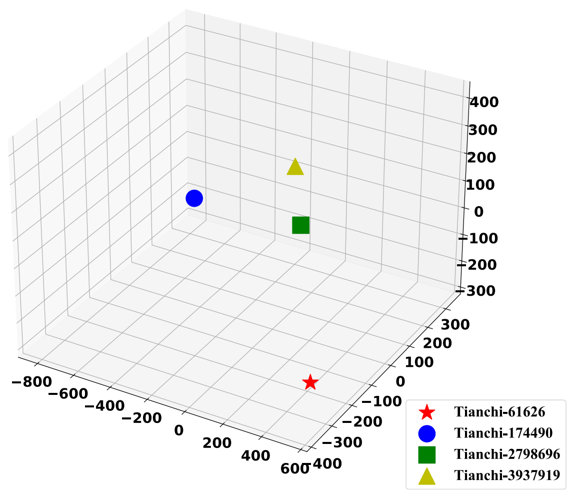
5.3.5.4 Visualization of Customized Models

To further explore the model adaptation, we visualize the customized models for the four downstream Tianchi datasets. Specifically, we extract the parameters of the first GNN layer of the

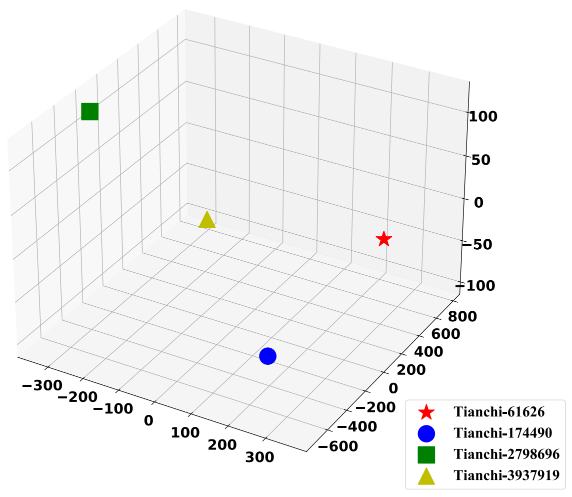
Table 5.7: Performances comparison of ADAPT with DRNL node labeling method and DeepWalk node labeling methods on Tianchi datasets with 40% as training.

	Tianchi-174490	Tianchi-61626	Tianchi-3937919	Tianchi-2798696
ADAPT-DRNL	60.07±3.20	52.39±1.27	58.79±3.37	48.66±2.86
ADAPT-DeepWalk	44.16±5.27	26.99±5.7	36.7±8.49	55.03±4.65

four customized models and then use t-sne [70] to project them to a 3-dimensional space. These 3-dimensional embeddings are then visualized in Figure 5.8, from which we can observe that the GNN adaptor indeed generates different customized models for different recommendation graphs. In addition, for graphs with similar graph properties, their corresponding customized models are more similar than those that are relatively different.



(a) gnn layer 1 weight.



(b) gnn layer 1 bias.

Figure 5.8: Visualization of customized models for different downstream graphs on Tianchi dataset.

5.3.5.5 Test on a larger dataset

We further test the proposed ADAPT on a relatively larger downstream dataset graph compared to the other downstream graphs. This dataset is from the romance category of MovieLens, which consists of 20407 users, 376 items, and 30496 edges. We compare the model performances with 60% edges as the training set. The result is shown in Table 5.8, where the proposed ADAPT method also achieves the best performance.

Table 5.8: Performance comparison on MovieLens-Romance dataset (with 20407 users, 376 items and 30496 edges) with 60% as training (The reported performance is measured by HR in %).

MF	NGCF	LightGCN	GCC	ADAPT-D	ADAPT-J
72.19±1.44	76.6±2.64	78.54±1.19	71.63±4.29	79.01±1.1	76.8±5.05

5.4 Related Work

Our work is related to graph neural networks, pre-training for graph neural networks, recommendation models based on graph neural network, inductive recommendation and cross-domain recommendation. Next, we briefly review representative methods from each category.

5.4.1 Graph Neural Networks

In recent years, increasing attention and efforts have been devoted into graph neural networks, which successfully extend deep neural networks to graph data. Graph neural networks are theoretically and empirically demonstrated to be very powerful in graph representation learning [52, 27], and have achieved great success in various applications from different domains, such as natural language processing [124, 43, 11], computer vision [83, 114] and recommendation [123, 125, 110, 96, 35, 117, 47, 137]. There are mainly two groups of graph neural networks: the spectral-based methods and the spatial-based methods. In [93], the first graph neural network is proposed from the spatial perspective to solve both graph and node level tasks, which aggregates information from neighboring nodes for each node in every layer. Subsequently, Bruna et al. [10] propose to generalize the convolution operation to the graph domain based on graph Laplacian theory from the spectral perspective. Following this work, Defferrard et al [13] propose ChebNet, which uses chebyshev polynomials to modulate the graph Fourier coefficients for different graph signals. Next, Kipf and Welling [52] further simplify ChebNet via some assumptions and propose graph convolutional networks (GCNs). It is remarkable that despite developed from the spectral perspective, GCNs can also be well illustrated in a spatial way. From then on, multiple spatial-based graph neural networks have been proposed. A comprehensive overview about GNNs can be found in recent surveys [119, 135] and books [68]. In addition, there emerge some work trying to further explore the rationale behind GNNs, such as Ma et al [67] propose that most existent GNNs can be unified as graph signal denoising. In addition to the aforementioned work mainly focusing on graph convolution operation, there are also numerous works targeting at graph pooling operation, which summarizes graph representation from node representations and plays an essential role in graph representation learning. There are simple pooling methods, such as directly averaging all the node representations as the graph representation [17] or adding a virtual node that connects to all the nodes in the graph and then taking its node representation as the graph representation [60]. An increasing number of hierarchical pooling methods have been proposed to learn graph representation hierarchically, which are believed to be able to better capture the graph structure information. Specifically, DiffPool [126] is proposed to learn a differentiable soft cluster assignment for each node at every GNN layer. Inspired by encoder-decoder model, graph U-Net [22] is designed to consist of graph pooling (gPool) and graph unpooling (gUnpool) operations, where gPool can adaptively select important nodes based on the importance scores computed on a learnable projection vector. Furthermore, RepPool [58] proposes a learnable way to better integrate non-selected nodes, which can better preserve the information of both the important nodes and normal nodes .In addition, EigenPooling [69] is designed from the perspective of graph Fourier transform and it can leverage both the node features and the local structures.

5.4.2 Pre-training for Graph Neural Networks

Inspired by the great success of pre-training techniques in multiple domains such as computer vision and natural language processing [72, 86, 15, 14, 32, 44, 45], many researchers have also started to explore how to a apply pre-training appropriately in GNN models [40, 84, 41, 65, 29]. Hu et al. [40] propose a pre-training strategy based on self-supervised learning, which focuses on pre-training a GNN model at both the node level and graph level, so that the pre-trained model can learn effective node representations and graph representations simultaneously. Qiu et al. propose GCC [84], a GNN pre-trainng framework based on contrastive learning, which aims at capturing transferable topological knowledge across multiple graphs.

GPT-GNN [41] is a GNN pre-training framework built on generative model. It aims at capturing both the structure information and the semantic information via generating node attributes and edges alternatively. In addition, some work focus on bridging the gap between GNN pre-training and GNN fine-tuning. For example, L2P-GNN [65] is proposed to mimic fine-tuning during the pre-training process via a dual-adaption mechanism at both the node level and graph level. Besides, there also emerges GNN pre-training work specially for alleviating the cold-start problem in recommendation tasks [29], where researchers propose to use embedding reconstruction as the pre-training task, so that the cold-start user/item can get a good representation from the reconstruction knowledge.

5.4.3 Graph Neural Networks for Recommendation Systems

Recommendation tasks are to predict a user's preference based on its historical interactions with various items. Historical user-item interactions can be directly denoted as a bipartite graph, and thus it is very natural to apply graph neural networks to recommendation tasks. In fact, there exist numerous works focusing on exploring GNNs for recommendation systems and many of them have achieved promising performance [125, 110, 35, 19, 109]. PinSage [125] is designed especially for recommendation tasks based on GraphSage, and can be directly applied to a web-scale recommendation tasks. For each user or item node, it utilizes a random-walk based sampling method to sample some neighboring nodes for information aggregation. NGCF [110] is designed to explicitly capture high-order connectivity in user-item interaction graphs via embedding propagation. Later on, He et al. propose LightGCN [35], which simplifies the design of GNNs especially for recommendation tasks via eliminating feature transformation and non-linear activation function, and has achieved significant performance improvement. Furthermore, Wu et al. [116] explore to improve the performance of GNN models for recommendations by incorporating self-supervised learning techniques. This work can be supplementary to most supervised GNN models for recommendations, which aim at improving user and item representations via self-discrimination. These GNN-based recommendation methods only relying on the user-item interaction graphs. There are also works focusing on utilizing GNNs to deal with side information, such as social networks and knowledge graphs, to facilitate recommendation performance. For examples, GraphRec [19] is proposed to use two graph attention networks to learn user embeddings and item embeddings, and the user embedding is learned from both the social graph and interaction graph. KGAT [109] is proposed to integrate the user-item interaction graph and the knowledge graph into one unified graph by viewing the user-item interaction as one type of the relations, and to use attentive embedding propagation layers to refine embeddings over this unified graph.

5.4.4 Inductive Recommendation and Cross-Domain Recommendation

Inductive recommendations focus on recommendation tasks in inductive scenarios, where there are emerging users or items that do not appear in the training process. Most methods [53, 36, 110, 35] that leverage one-hot identification representation as input to learn user/item embeddings that typically cannot be applied in inductive scenarios. Some item-based methods [92, 48] aim at making predictions based on similarities between the new items and a fixed item set, which can recommend existent items to new users. Recently, there are also a few models proposed especially for inductive recommendation where there are both new users and items. IGCN [118] is one of such methods. It initiates the embeddings of the new users and items by leveraging the embeddings of some pre-defined core items/users and global item/user representation. CF-GCN [87] learns embeddings of the new users and items via fusing embeddings of existent users/items of different layers. Most of these methods aim at learning representations for new items/users via their interactions with items/users seen in the training phase. The aforementioned inductive scenario is quite different from the pre-training scenario we discuss about at this work. The inductive scenario aims at handling new items/users in an existent recommendation graph, while the proposed pre-training scenario targets at transferring common knowledge among different recommendation graphs. Furthermore, the methods [92, 48, 118, 87] for the inductive scenario are not applicable in the proposed pre-training scenario, since it requires interactions between new items/users between the core items/users seen in the training phase, but there is no such common core items/users among different recommendation graphs in the proposed pre-training scenario.

Cross-domain recommendation is also proposed to alleviate the data scarcity problem in recommendation in sparser domains via transferring knowledge from richer domains. There are different scenarios in cross-domain recommendation [140], among which single-target scenario is most similar with the proposed pre-training scenario, where richer data from source domains are leveraged to help recommendation in a target domain with sparser data. In this scenario, it is assumed that there are common users or items among source domains and the target domain [131, 139, 61], or there exist attributes of the same modality among these domains, such as user ratings [8] and semantic properties [56, 130]. Typical techniques adopted in cross-domain recommendation include transfer learning [73, 136, 34], domain adaptation [49, 127], multi-task learning [4, 64] and so on. Although sharing similar goals, cross-domain recommendation and the proposed pre-training recommendation are quite different. In the proposed pre-training recommendation scenario, there are no common users or items among different domains. Also, no user/item attributes of the same modality are provided. Therefore, most cross-domain recommendation methods are not applicable to the studied scenario.

5.5 Conclusion

In this paper, we propose an adaptive graph pre-training framework for localized collaborative filtering, ADAPT, which can effectively help alleviate the data scarcity challenge in recommendation tasks. There are two key components in the proposed ADAPT: the meta-LGNN and the GNN adaptor. Specifically, the meta-LGNN is a novel GNN-based recommendation method from a new perspective of local structure. It aims at encoding the collaborative filtering information into the graph representation for the neighboring structure of a given user-item pair, and it does not require learning user/item embeddings. Thus it is more flexible for pre-training compared to existing GNN-based recommendation methods. The GNN adaptor is the key to allowing the effectiveness of taking advantage of multiple graphs for pre-training. It can capture the difference for each graph, and adapt the meta-LGNN model to the customized GNN model accordingly. Overall, the proposed ADAPT is able to transfer common knowledge from the pre-training graphs for the target downstream recommendation graph and to capture their uniqueness in the meanwhile. Extensive experiments have validated the effectiveness and superiority of the proposed ADAPT in GNN pre-training for recommendations.

There are a few directions we can further explore in the future. First, the current implementation of the meta-LGN is based on GCN, which is a vanilla model in graph representation learning. The performance of the meta-LGN could be further improved by replacing the GCN model with other advanced GNN models such as GIN. Second, the node labeling method currently adopted by the meta-LGN treats user and item nodes equally. In further exploration, we may consider distinguishing the user and item nodes in the node labeling method. Third, the designs of the adaptor model and adaptation operation are also flexible, and there are some alternative models and operations that can further be explored on the proposed framework. Last but not least, instead of pre-training the proposed framework on numerous independent user-item interaction graphs, we may consider how to leverage some available shared user or item nodes among these graphs.

CHAPTER 6

CONCLUSION

Graph Neural Networks have been proven effective in graph representation learning, and have achieved revolutionary achievements in many graph-related tasks, including graph classification, node classification, and link prediction. GNNs have aroused great research interests and many novel GNN models have been designed and applied in various applications. While significant progress has been made in regard to GNNs, their out-of-distribution generalization problem has not been effectively solved and this hinders their effective applications in real-world scenarios. In this dissertation, we focus on the out-of-distribution generalization in GNNs and propose to improve their out-of-distribution generalization ability from two perspectives, i.e., a novel training perspective and an advanced learning perspective. In addition, we aim at addressing the data sparsity challenge in applying GNNs in the recommendation tasks and improving the abilities of GNNs in terms of generalization and knowledge transfer in the real-world application of recommendations. In this chapter, we summarize our proposed methods and discuss some promising future research directions.

6.1 Summary

To bridge the performance gap of GNNs between the training set and the test set, we propose to design a novel training paradigm, the test-time training paradigm, for GNNs. In other words, the GNN models are further trained for the test graph samples in the test time. Specifically, we propose the first test-time training framework for GNNs (GT3) to improve the out-of-distribution generalization ability of GNNs in the graph classification task. A two-level self-supervised learning task is carefully designed to adjust the GNN model for each test graph sample. Also, we propose an additional adaptation constraint to prevent undesirable excessive feature distortion during test-time training. Extensive experiments have been conducted to validate the effectiveness of the proposed GT3, and we also provide a theoretical analysis to demonstrate the benefits of test-time training for GNNs.

Apart from the training perspective, we also propose Customized-GNN, which aims at improving GNNs' out-of-distribution generalization ability from a novel learning perspective. Specifically, we first explore the graph data diversity in terms of graph structure and then investigate the influence of the data diversity on GNNs' performance. Inspired by the investigation, we propose to consider the diverse graph structure properties in GNNs' training process. Specifically, the proposed Customized-GNN can generate a sample-specific GNN for each graph sample based on its graph structure. The effectiveness of Customized-GNN is validated via comprehensive experiments. In addition, the proposed framework is very general and flexible and thus can be applied to numerous existing GNN models.

In addition to improving the GNNs' out-of-distribution generalization ability via two general frameworks, we also focus on addressing the challenges of applying GNNs in the real-world application of recommendations. More specifically, we aim at improving the abilities of GNNs in terms of generalization and knowledge transfer in the recommendation scenario. To achieve this goal, we first propose a novel GNN-based recommendation method, LGCF, which is able to learn the recommendation knowledge from the localized graph induced from a target user and item. Unlike most other GNN-based recommendation methods, LGCF does not require learning effective embeddings for each user and item. Thus, it is essentially more friendly in the recommendation scenarios of data sparsity, and more importantly, it paves a way to transfer recommendation knowledge among different recommendation scenarios. Next, based on LGCF, we further design an adaptive pre-training framework, ADAPT, which is able to transfer common collaborative filtering knowledge among different user-item graphs and preserve the uniqueness of each graph simultaneously.

6.2 Future Directions

Although GNNs have achieved promising progress in numerous graph-related tasks, the out-of-distribution generalization of GNNs is a vital problem that calls for more dedication and exploration from different perspectives. We discuss some possible future research directions as follows:

- **Generalization challenges brought by node diversity:** The two methods we designed in this dissertation mainly focus on the graph-level tasks where the training graph samples and the test graph samples are diverse in terms of the graph structure. The scenario where the distributions of node attributes vary between the training set and test set has not been considered yet. However, this scenario is also quite common in real-world applications. For example, in social networks, different social platforms may collect different demographic information from their users. Thus, to further solve the out-of-distribution generalization problem of GNNs in real-world applications, it is necessary to take the distribution difference of node attributes into consideration, instead of only considering the graph structure diversity. Also, it is observed that some node properties such as degree and local subgraph surrounding the target node play important roles in node-level tasks [103, 42]. Therefore, it is also promising to take node property diversity into consideration to further improve the out-of-distribution generalization ability of GNNs.

- **Generalization challenges in different applications:** There exist various unique challenges and diverse domain knowledge for GNNs in different applications. In this dissertation, we focus on improving the GNNs' performance and out-of-distribution generalization ability for the recommendation task. However, there are many other applications, such as molecular prediction and single-cell analysis. In order to fulfill GNNs' potential in these applications, we need to target at their unique challenges and consider their domain knowledge to further improve the out-of-distribution generalization ability of GNNs in various applications.

BIBLIOGRAPHY

[1] Movielens. https://grouplens.org/datasets/movielens/. Accessed May 23, 2021.

[2] User behavior data from taobao for recommendation. https://tianchi.aliyun.com/dataset/dataDetail?dataId=649&userId=1. Accessed May 23, 2021.

[3] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, 17(6):734–749, 2005.

[4] Deepak Agarwal, Bee-Chung Chen, and Bo Long. Localized factor models for multi-context recommendation. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 609–617, 2011.

[5] Charu C Aggarwal. Model-based collaborative filtering. In *Recommender systems*, pages 71–138. Springer, 2016.

[6] Alexander Bartler, Andre Bühler, Felix Wiewel, Mario Döbler, and Bin Yang. Mt3: Meta test-time training for self-supervised test-time adaption. *arXiv preprint arXiv:2103.16201*, 2021.

[7] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[8] Shlomo Berkovsky, Tsvi Kuflik, and Francesco Ricci. Cross-domain mediation in collaborative filtering. In *International Conference on User Modeling*, pages 355–359. Springer, 2007.

[9] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56, 2005.

[10] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[11] Deng Cai and Wai Lam. Graph transformer for graph-to-sequence learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7464–7471, 2020.

[12] Ming Chen, Zhewei Wei, Bolin Ding, Yaliang Li, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. Scalable graph neural networks via bidirectional propagation. *Advances in neural information processing systems*, 33:14556–14566, 2020.

[13] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

[14] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[16] Paul D Dobson and Andrew J Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 330(4):771–783, 2003.

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

[18] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. *arXiv preprint arXiv:1912.09893*, 2019.

[19] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The World Wide Web Conference*, pages 417–426, 2019.

[20] Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. Splinecnn: Fast geometric deep learning with continuous b-spline kernels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 869–877, 2018.

[21] Yang Fu, Sifei Liu, Umar Iqbal, Shalini De Mello, Humphrey Shi, and Jan Kautz. Learning to track instances without video annotations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8680–8689, 2021.

[22] Hongyang Gao and Shuiwang Ji. Graph u-nets. In *international conference on machine learning*. PMLR, 2019.

[23] Hongyang Gao, Yi Liu, and Shuiwang Ji. Topology-aware graph pooling networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[24] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017.

[25] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: a factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247*, 2017.

[26] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

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

[28] Nicklas Hansen, Rishabh Jangir, Yu Sun, Guillem Alenyà, Pieter Abbeel, Alexei A Efros, Lerrel Pinto, and Xiaolong Wang. Self-supervised policy adaptation during deployment. *arXiv preprint arXiv:2007.04309*, 2020.

[29] Bowen Hao, Jing Zhang, Hongzhi Yin, Cuiping Li, and Hong Chen. Pre-training graph neural networks for cold-start users and items representation. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pages 265–273, 2021.

[30] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4), 2015.

[31] Kaveh Hassani and Amir Hosein Khasahmadi. Contrastive multi-view representation learning on graphs. In *International Conference on Machine Learning*, pages 4116–4126. PMLR, 2020.

[32] Kaiming He, Ross Girshick, and Piotr Dollár. Rethinking imagenet pre-training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4918–4927, 2019.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[34] Ming He, Jiuling Zhang, Peng Yang, and Kaisheng Yao. Robust transfer learning for cross-domain collaborative filtering using multiple rating patterns approximation. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 225–233, 2018.

[35] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 639–648, 2020.

[36] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182, 2017.

[37] Thomas Hofmann. Latent semantic models for collaborative filtering. *ACM Transactions on Information Systems (TOIS)*, 22(1):89–115, 2004.

[38] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[39] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *NeurIPS*, 2020.

[40] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*, 2019.

[41] Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. Gpt-gnn: Generative pre-training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1857–1867, 2020.

[42] K. Huang and M. Zitnik. Graph meta learning via local subgraphs. *arXiv e-prints*, 2020.

[43] Lianzhe Huang, Dehong Ma, Sujian Li, Xiaodong Zhang, and Houfeng Wang. Text level graph neural network for text classification. *arXiv preprint arXiv:1910.02356*, 2019.

[44] Minyoung Huh, Pulkit Agrawal, and Alexei A Efros. What makes imagenet good for transfer learning? *arXiv preprint arXiv:1608.08614*, 2016.

[45] Xiaodong Jiang, Ronghang Zhu, Sheng Li, and Pengsheng Ji. Co-embedding of nodes and edges with graph neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[46] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. Graph structure learning for robust graph neural networks. *arXiv preprint arXiv:2005.10203*, 2020.

[47] Yuanyuan Jin, Wei Zhang, Xiangnan He, Xinyu Wang, and Xiaoling Wang. Syndrome-aware herb recommendation with multi-graph convolution network. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*, pages 145–156. IEEE, 2020.

[48] Santosh Kabbur, Xia Ning, and George Karypis. Fism: factored item similarity models for top-n recommender systems. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 659–667, 2013.

[49] Heishiro Kanagawa, Hayato Kobayashi, Nobuyuki Shimizu, Yukihiro Tagami, and Taiji Suzuki. Cross-domain recommendation via deep domain adaptation. In *European Conference on Information Retrieval*, pages 20–29. Springer, 2019.

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

[51] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *ArXiv preprint*, 2016.

[52] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[53] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[54] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International Conference on Machine Learning*, pages 3519–3529. PMLR, 2019.

[55] Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura Barnes, and Donald Brown. Text classification algorithms: A survey. *Information*, 10(4):150, 2019.

[56] Anil Kumar, Nitesh Kumar, Muzammil Hussain, Santanu Chaudhury, and Sumeet Agarwal. Semantic clustering-based cross-domain recommendation. In *2014 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, pages 137–141. IEEE, 2014.

[57] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. *arXiv preprint arXiv:1904.08082*, 2019.

[58] Juanhui Li, Yao Ma, Yiqi Wang, Charu Aggarwal, Chang-Dong Wang, and Jiliang Tang. Graph pooling with representativeness. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 302–311. IEEE, 2020.

[59] Ruoyu Li, Sheng Wang, Feiyun Zhu, and Junzhou Huang. Adaptive graph convolutional neural networks. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

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

[61] Meng Liu, Jianjun Li, Guohui Li, and Peng Pan. Cross domain recommendation via bi-directional transfer graph collaborative filtering networks. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 885–894, 2020.

[62] Xiaorui Liu, Wei Jin, Yao Ma, Yaxin Li, Hua Liu, Yiqi Wang, Ming Yan, and Jiliang Tang. Elastic graph neural networks. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, Proceedings of Machine Learning Research, 2021.

[63] Yuejiang Liu, Parth Kothari, Bastien van Delft, Baptiste Bellot-Gurlet, Taylor Mordan, and Alexandre Alahi. Ttt++: When does self-supervised test-time training fail or thrive? *Advances in Neural Information Processing Systems*, 34, 2021.

[64] Yichao Lu, Ruihai Dong, and Barry Smyth. Why i like it: multi-task learning for recommendation and explanation. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 4–12, 2018.

[65] Yuanfu Lu, Xunqiang Jiang, Yuan Fang, and Chuan Shi. Learning to pre-train graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 4276–4284, 2021.

[66] Dongsheng Luo, Wei Cheng, Wenchao Yu, Bo Zong, Jingchao Ni, Haifeng Chen, and Xiang Zhang. Learning to drop: Robust graph neural network via topological denoising. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pages 779–787, 2021.

[67] Yao Ma, Xiaorui Liu, Tong Zhao, Yozen Liu, Jiliang Tang, and Neil Shah. A unified view on graph neural networks as graph signal denoising. *arXiv preprint arXiv:2010.01777*, 2020.

[68] Yao Ma and Jiliang Tang. *Deep learning on graphs*. Cambridge University Press, 2021.

[69] Yao Ma, Suhang Wang, Charu C Aggarwal, and Jiliang Tang. Graph convolutional networks with eigenpooling. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 723–731, 2019.

[70] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9:2579–2605, 2008.

[71] Julian McAuley. Amazon product data. http://jmcauley.ucsd.edu/data/amazon/. Accessed May 23, 2021.

[72] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[73] Orly Moreno, Bracha Shapira, Lior Rokach, and Guy Shani. Talmud: transfer learning for multiple domains. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 425–434, 2012.

[74] Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*, 2020.

[75] Vaishnavh Nagarajan, Anders Andreassen, and Behnam Neyshabur. Understanding the failure modes of out-of-distribution generalization. *arXiv preprint arXiv:2010.15775*, 2020.

[76] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[77] Ali Bou Nassif, Ismail Shahin, Imtinan Attili, Mohammad Azzeh, and Khaled Shaalan. Speech recognition using deep neural networks: A systematic review. *IEEE Access*, 7:19143–19165, 2019.

[78] Shumpei Okura, Yukihiro Tagami, Shingo Ono, and Akira Tajima. Embedding-based news recommendation for millions of users. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1933–1942, 2017.

[79] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *ArXiv preprint*, 2018.

[80] Karl Pearson. The problem of the random walk. *Nature*, 72(1867):342–342, 1905.

[81] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[82] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD*, pages 701–710, 2014.

[83] Siyuan Qi, Wenguan Wang, Baoxiong Jia, Jianbing Shen, and Song-Chun Zhu. Learning human-object interactions by graph parsing neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 401–417, 2018.

[84] Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. Gcc: Graph contrastive coding for graph neural network pre-training. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1150–1160, 2020.

[85] Xipeng Qiu, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai, and Xuanjing Huang. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*, pages 1–26, 2020.

[86] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

[87] Rahul Ragesh, Sundararajan Sellamanickam, Vijay Lingam, Arun Iyer, and Ramakrishna Bairi. User embedding based neighborhood aggregation method for inductive recommendation. 2021.

[88] Ekagra Ranjan, Soumya Sanyal, and Partha Pratim Talukdar. Asap: Adaptive structure aware pooling for learning hierarchical graph representations. *arXiv preprint arXiv:1911.07979*, 2019.

[89] Waseem Rawat and Zenghui Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, 29(9):2352–2449, 2017.

[90] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618*, 2012.

[91] Yu Rong, Yatao Bian, Tingyang Xu, Weiyang Xie, Ying Wei, Wenbing Huang, and Junzhou Huang. Self-supervised graph transformer on large-scale molecular data. In *NeurIPS*, 2020.

[92] B. Sarwar. Item-based collaborative filtering recommendation algorithms. In *Proc. the 10th International World Wide Web Conference (WWW10), Hong Kong, May 1-5 (2001)*, 2001.

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

[94] Kristof Schütt, Pieter-Jan Kindermans, Huziel Enoc Sauceda Felix, Stefan Chmiela, Alexandre Tkatchenko, and Klaus-Robert Müller. Schnet: A continuous-filter convolutional neural network for modeling quantum interactions. In *Advances in neural information processing systems*, pages 991–1001, 2017.

[95] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.

[96] Heng-Shiou Sheu and Sheng Li. Context-aware graph embedding for session-based news recommendation. In *Fourteenth ACM conference on recommender systems*, pages 657–662, 2020.

[97] Tzu-An Song, Samadrita Roy Chowdhury, Fan Yang, Heidi Jacobs, Georges El Fakhri, Quanzheng Li, Keith Johnson, and Joyita Dutta. Graph convolutional neural networks for alzheimer's disease classification. In *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)*, pages 414–417. IEEE, 2019.

[98] Xiaoyuan Su and Taghi M Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009, 2009.

[99] Fan-Yun Sun, Jordan Hoffmann, Vikas Verma, and Jian Tang. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. *arXiv preprint arXiv:1908.01000*, 2019.

[100] Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *International Conference on Machine Learning*, pages 9229–9248. PMLR, 2020.

[101] Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei A Efros, and Moritz Hardt. Test-time training for out-of-distribution generalization. 2019.

[102] Qiaoyu Tan, Ninghao Liu, and Xia Hu. Deep representation learning for social network analysis. *Frontiers in big Data*, 2:2, 2019.

[103] Xianfeng Tang, Huaxiu Yao, Yiwei Sun, Yiqi Wang, Jiliang Tang, Charu Aggarwal, Prasenjit Mitra, and Suhang Wang. Investigating and mitigating degree-related biases in graph convoltuional networks. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 1435–1444, 2020.

[104] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. Fast random walk with restart and its applications. In *Sixth international conference on data mining (ICDM'06)*, pages 613–622. IEEE, 2006.

[105] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. Composition-based multi-relational graph convolutional networks. In *International Conference on Learning Representations*, 2020.

[106] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[107] Petar Velickovic, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. *ICLR (Poster)*, 2(3):4, 2019.

[108] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. Collaborative deep learning for recommender systems. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1235–1244, 2015.

[109] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. Kgat: Knowledge graph attention network for recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 950–958, 2019.

[110] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, pages 165–174, 2019.

[111] Xiaoyang Wang, Yao Ma, Yiqi Wang, Wei Jin, Xin Wang, Jiliang Tang, Caiyan Jia, and Jian Yu. Traffic flow prediction via spatial temporal graph neural network. In *Proceedings of The Web Conference 2020*, pages 1082–1092, 2020.

[112] Yiqi Wang, Chaozhuo Li, Mingzheng Li, Wei Jin, Yuming Liu, Hao Sun, Xing Xie, and Jiliang Tang. Localized graph collaborative filtering. In *Proceedings of the 2022 SIAM International Conference on Data Mining (SDM)*, pages 540–548. SIAM, 2022.

[113] Yiqi Wang, Chaozhuo Li, Zheng Liu, Mingzheng Li, Jiliang Tang, Xing Xie, Lei Chen, and Philip S Yu. An adaptive graph pre-training framework for localized collaborative filtering. *ACM Transactions on Information Systems (TOIS)*, 2022.

[114] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12, 2019.

[115] Felix Wu, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks. In *ICML*, Proceedings of Machine Learning Research, 2019.

[116] Jiancan Wu, Xiang Wang, Fuli Feng, Xiangnan He, Liang Chen, Jianxun Lian, and Xing Xie. Self-supervised graph learning for recommendation. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 726–735, 2021.

[117] Le Wu, Lei Chen, Pengyang Shao, Richang Hong, Xiting Wang, and Meng Wang. Learning fair representations for recommendation: A graph-based perspective. In Jure Leskovec, Marko Grobelnik, Marc Najork, Jie Tang, and Leila Zia, editors, *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, pages 2198–2208. ACM / IW3C2, 2021.

[118] Yunfan Wu, Qi Cao, Huawei Shen, Shuchang Tao, and Xueqi Cheng. Inductive representation based graph convolution network for collaborative filtering. 2021.

[119] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 2020.

[120] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

[121] Hong-Jian Xue, Xinyu Dai, Jianbing Zhang, Shujian Huang, and Jiajun Chen. Deep matrix factorization models for recommender systems. In *IJCAI*, volume 17, pages 3203–3209. Melbourne, Australia, 2017.

[122] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1365–1374. ACM, 2015.

[123] Jheng-Hong Yang, Chih-Ming Chen, Chuan-Ju Wang, and Ming-Feng Tsai. Hop-rec: high-order proximity for implicit recommendation. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 140–144, 2018.

[124] Liang Yao, Chengsheng Mao, and Yuan Luo. Graph convolutional networks for text classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 7370–7377, 2019.

[125] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *KDD*, 2018.

[126] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*, pages 4800–4810, 2018.

[127] Feng Yuan, Lina Yao, and Boualem Benatallah. Darec: Deep domain adaptation for cross-domain recommendation via transferring rating patterns. *arXiv preprint arXiv:1905.10760*, 2019.

[128] Hao Yuan and Shuiwang Ji. Structpool: Structured graph pooling via conditional random fields. In *International Conference on Learning Representations*, 2020.

[129] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *arXiv preprint arXiv:1802.09691*, 2018.

[130] Qian Zhang, Peng Hao, Jie Lu, and Guangquan Zhang. Cross-domain recommendation with semantic correlation in tagging systems. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019.

[131] Qian Zhang, Jie Lu, Dianshuang Wu, and Guangquan Zhang. A cross-domain recommender system with kernel-induced knowledge transfer for overlapping entities. *IEEE transactions on neural networks and learning systems*, 30(7):1998–2012, 2018.

[132] Yian Zhang, Alex Warstadt, Haau-Sing Li, and Samuel R Bowman. When do you need billions of words of pretraining data? *arXiv preprint arXiv:2011.04946*, 2020.

[133] Zhen Zhang, Jiajun Bu, Martin Ester, Jianfeng Zhang, Chengwei Yao, Zhi Yu, and Can Wang. Hierarchical graph pooling with structure learning. *arXiv preprint arXiv:1911.05954*, 2019.

[134] Zhen Zhang, Hongxia Yang, Jiajun Bu, Sheng Zhou, Pinggang Yu, Jianwei Zhang, Martin Ester, and Can Wang. Anrl: Attributed network representation learning via deep neural networks. In *IJCAI*, volume 18, pages 3155–3161, 2018.

[135] Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[136] Lili Zhao, Sinno Jialin Pan, Evan Wei Xiang, Erheng Zhong, Zhongqi Lu, and Qiang Yang. Active transfer learning for cross-system recommendation. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[137] Yu Zheng, Chen Gao, Xiangnan He, Yong Li, and Depeng Jin. Price-aware recommendation with graph convolutional networks. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*, pages 133–144. IEEE, 2020.

[138] Kaixiong Zhou, Qingquan Song, Xiao Huang, Daochen Zha, Na Zou, and Xia Hu. Multi-channel graph neural networks. *arXiv preprint arXiv:1912.08306*, 2019.

[139] Feng Zhu, Yan Wang, Chaochao Chen, Guanfeng Liu, and Xiaolin Zheng. A graphical and attentional framework for dual-target cross-domain recommendation. In *IJCAI*, pages 3001–3008, 2020.

[140] Feng Zhu, Yan Wang, Chaochao Chen, Jun Zhou, Longfei Li, and Guanfeng Liu. Cross-domain recommendation: challenges, progress, and prospects. *arXiv preprint arXiv:2103.01696*, 2021.

[141] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Graph contrastive learning with adaptive augmentation. In *Proceedings of the Web Conference 2021*, pages 2069–2080, 2021.

APPENDIX A

TEST-TIME TRAINING FOR GRAPH NEURAL NETWORKS

A.1 Theorem Proof

A.1.1 Proof of Theorem 1

Denote $\mathcal{L}(y;\mathbf{g}) = -\sum_{c=1}^{C} 1_{y=c}\log(\frac{e^{\mathbf{g}_c}}{\sum_{i=1}^{C} e^{\mathbf{g}_i}})$, and from Eq 2.15, we have

$$\mathbf{g} = f_{\text{GNN}}(\mathbf{A},\mathbf{X};\boldsymbol{\theta}) = \mathbf{1^T A^L X}\boldsymbol{\theta}. \tag{A.1}$$

$$\begin{aligned}\mathcal{L}(y;\mathbf{g}) &= -\sum_{c=1}^{C} 1_{y=c}\log(\frac{e^{\mathbf{g}_c}}{\sum_{i=1}^{C} e^{\mathbf{g}_i}}) \\ &= -\log(\frac{e^{\mathbf{z_c}}}{\sum_{i=1}^{C} e^{\mathbf{z_i}}}).\end{aligned} \tag{A.2}$$

The Hessian Matrix of Eq A.2, we have

$$\mathbf{H}_{ij} = \begin{cases} -\mathbf{p}_i\mathbf{p}_j & \text{i} \neq \text{j} \\ \mathbf{p}_i - \mathbf{p}_i^2 & \text{i=j}, \end{cases} \tag{A.3}$$

where $\mathbf{p}_i$ is defined in Eq 2.16.

For $\forall \mathbf{a} \in \mathbb{R}^C$, we have

$$\begin{aligned}\mathbf{a}^T\mathbf{H}\mathbf{a} &= \sum_{i=1}^{C}\mathbf{p}_i\mathbf{a}_i^2 - (\sum_{i=1}^{C}\mathbf{a}_i\mathbf{p}_i)^2 \\ &= \sum_{i=1}^{C}\mathbf{p}_i\mathbf{a}_i^2 - (\sum_{i=1}^{C}\mathbf{a}_i\sqrt{\mathbf{p}_i}\sqrt{\mathbf{p}_i})^2\end{aligned} \tag{A.4}$$

According to Cauchy-Schwarz Inequality, we have

$$(\sum_{i=1}^{C}\mathbf{a}_i\sqrt{\mathbf{p}_i}\sqrt{\mathbf{p}_i})^2 \leq (\sum_{i=1}^{C}\mathbf{a}_i{}^2\mathbf{p}_i)(\sum_{i=1}^{C}\mathbf{p}_i). \tag{A.5}$$

Thus, we have

$$\mathbf{a}^T\mathbf{H}\mathbf{a} \geq \sum_{i=1}^{C}\mathbf{p}_i\mathbf{a}_i^2 - (\sum_{i=1}^{C}\mathbf{a}_i{}^2\mathbf{p}_i)(\sum_{i=1}^{C}\mathbf{p}_i). \tag{A.6}$$

According to the definition of $\mathbf{p}_i$ in Eq 2.16, we have

$$\sum_{i=1}^{C} \mathbf{p}_i = 1. \tag{A.7}$$

Therefore,

$$\mathbf{a}^T \mathbf{H} \mathbf{a} \geq 0. \tag{A.8}$$

Thus, we have proved that the Hessian Matrix of the function defined in Eq A.2, i.e., **H**, is positive semi-definite (PSD). This is equivalent to prove that $\mathcal{L}(y; \mathbf{g})$ is **convex** in **g** (**property 1**).

For $\forall \mathbf{b} \in \mathbb{R}^C$ and $||\mathbf{b}|| = 1$, we have

$$\begin{aligned} \mathbf{b}^T \mathbf{H} \mathbf{b} &= \sum_{i=1}^{C} \mathbf{p}_i \mathbf{b}_i^2 - (\sum_{i=1}^{C} \mathbf{b}_i \mathbf{p}_i)^2 \\ &= \sum_{i=1}^{C} \mathbf{p}_i \mathbf{b}_i^2 - (\sum_{i=1}^{C} \mathbf{b}_i \sqrt{\mathbf{p}_i} \sqrt{\mathbf{p}_i})^2 \\ &\leq \sum_{i=1}^{C} \mathbf{b}_i^2 + (\sum_{i=1}^{C} \mathbf{b}_i \sqrt{\mathbf{p}_i} \sqrt{\mathbf{p}_i})^2. \end{aligned} \tag{A.9}$$

According to Cauchy-Schwarz Inequality, we can arrive at

$$\begin{aligned} \mathbf{b}^T \mathbf{H} \mathbf{b} &\leq \sum_{i=1}^{C} \mathbf{b}_i^2 + (\sum_{i=1}^{C} \mathbf{b}_i \sqrt{\mathbf{p}_i} \sqrt{\mathbf{p}_i})^2 \\ &\leq \sum_{i=1}^{C} \mathbf{b}_i^2 + (\sum_{i=1}^{C} \mathbf{b}_i^2)(\sum_{i=1}^{C} \mathbf{p}_i). \end{aligned} \tag{A.10}$$

Since $||\mathbf{b}|| = 1$ and $\sum_{i=1}^{C} \mathbf{p}_i = 1$, we have

$$\mathbf{b}^T \mathbf{H} \mathbf{b} \leq 2. \tag{A.11}$$

We have thus proved that the eigenvalues of the Hessian Matrix **H** are smaller than 2. This is equivalent to prove that $\mathcal{L}(y; \mathbf{g})$ is β**-smooth** in **g** (**property 2**).

By computing the first-order gradient of $\mathcal{L}(y; \mathbf{g})$ over **z**, we have

$$\nabla_{\mathbf{g}} \mathcal{L}(\mathbf{A}, \mathbf{X}, y; \mathbf{g}) = \begin{cases} -1 + \mathbf{p}_i & \text{i=c} \\ \mathbf{p}_i & \text{i} \neq \text{c}. \end{cases} \tag{A.12}$$

Then, we can derive its L_2 norm

$$\begin{aligned} \|\nabla_{\mathbf{g}}\mathcal{L}(y;\mathbf{g})\| &= \sqrt{(-1+\mathbf{p}_c)^2 + \sum_{i\neq c}(\mathbf{p}_i)^2} \\ &= \sqrt{\sum_{i\neq c}(\mathbf{p}_i)^2 + \sum_{i\neq c}(\mathbf{p}_i)^2} \\ &\leq 2. \end{aligned} \tag{A.13}$$

We have thus proved that the L_2 norm of the first-order gradient of $\mathcal{L}(y;\mathbf{g})$ over $\mathbf{z}$ is bounded by the positive constant (**property 3**).

So far, we have proved that for all $\mathbf{g}, y$, $\mathcal{L}(y;\mathbf{g})$ is convex and β-smooth in $\mathbf{g}$, and both $\|\nabla_{\mathbf{g}}\mathcal{L}(y;\mathbf{g})\| \leq G$ for all $\mathbf{g}$, where G is a positive constant. According to Eq A.1, f_{GNN} is a linear transformation mapping function and won't change these three properties. Therefore, the proof of Theorem 1 is completed.

A.1.2 Proof of Theorem 2

We follow [101] to prove Theorem 2. For any η, by β-smoothness, we have

$$\begin{aligned} &\mathcal{L}_m(x,y;\boldsymbol{\theta}(x)) = \mathcal{L}_m(x,y;\boldsymbol{\theta}(x) - \eta\nabla_{\boldsymbol{\theta}}\mathcal{L}_s(x;\boldsymbol{\theta})) \\ &\leq \mathcal{L}_m(x,y;\boldsymbol{\theta}(x) - \eta\,\langle\nabla_{\boldsymbol{\theta}}\mathcal{L}_m(x,y;\boldsymbol{\theta}), \nabla_{\boldsymbol{\theta}}\mathcal{L}_s(x;\boldsymbol{\theta})\rangle \\ &+ \frac{\eta^2\beta}{2}\|\nabla_{\boldsymbol{\theta}}\mathcal{L}_s(x;\boldsymbol{\theta})\|^2. \end{aligned} \tag{A.14}$$

Denote

$$\eta^* = \frac{\langle\nabla_{\boldsymbol{\theta}}\mathcal{L}_m(x,y;\boldsymbol{\theta}), \nabla_{\boldsymbol{\theta}}\mathcal{L}_s(x;\boldsymbol{\theta})\rangle}{\beta\|\nabla_{\boldsymbol{\theta}}\mathcal{L}_s(x;\boldsymbol{\theta})\|^2}, \tag{A.15}$$

then we have

$$\begin{aligned} &\mathcal{L}_m(x,y;\boldsymbol{\theta} - \eta^*\nabla_{\boldsymbol{\theta}}\mathcal{L}_s(x;\boldsymbol{\theta})) \\ &\leq \mathcal{L}_m(x,y;\boldsymbol{\theta}) - \frac{\langle\nabla_{\boldsymbol{\theta}}\mathcal{L}_m(x,y;\boldsymbol{\theta}), \nabla_{\boldsymbol{\theta}}\mathcal{L}_s(x;\boldsymbol{\theta})\rangle^2}{2\beta\|\nabla_{\boldsymbol{\theta}}\mathcal{L}_s(x;\boldsymbol{\theta})\|^2)}. \end{aligned} \tag{A.16}$$

By the assumptions on the L_2 norm of the first-order gradients for the main task and the SSL task and the assumption on their inner product, we have

$$\mathcal{L}_m(x, y; \boldsymbol{\theta}) - \mathcal{L}_m(x, y; \boldsymbol{\theta} - \eta^* \nabla_{\boldsymbol{\theta}} \mathcal{L}_s(x; \boldsymbol{\theta})) \geq \frac{\epsilon^2}{2\beta G^2}. \tag{A.17}$$

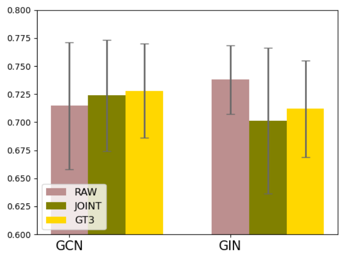
Denote $\eta = \frac{\epsilon}{\beta G^2}$, based on the assumptions, we have $0 < \eta \leq \eta^*$. Denote $\mathbf{t} = \nabla_{\boldsymbol{\theta}} \mathcal{L}_s(x; \boldsymbol{\theta})$, by convexity of $\mathcal{L}_m$,

$$\begin{aligned}
&\mathcal{L}_m(x, y; \boldsymbol{\theta}(x)) = \mathcal{L}_m(x, y; \boldsymbol{\theta} - \eta \mathbf{t}) \\
&= \mathcal{L}_m(x, y; (1 - \frac{\eta}{\eta^*})\boldsymbol{\theta} + \frac{\eta}{\eta^*}(\boldsymbol{\theta} - \eta^* \mathbf{t})) \\
&\leq (1 - \frac{\eta}{\eta^*})\mathcal{L}_m(x, y; \boldsymbol{\theta}) + \frac{\eta}{\eta^*}\mathcal{L}_m(x, y; \boldsymbol{\theta} - \eta^* \mathbf{t}) \\
&\leq (1 - \frac{\eta}{\eta^*})\mathcal{L}_m(x, y; \boldsymbol{\theta}) + \frac{\eta}{\eta^*}(\mathcal{L}_m(x, y; \boldsymbol{\theta}) - \frac{\epsilon^2}{2\beta G^2}) \\
&= \mathcal{L}_m(x, y; \boldsymbol{\theta}) - \frac{\eta}{\eta^*}\frac{\epsilon^2}{2\beta G^2}.
\end{aligned} \tag{A.18}$$

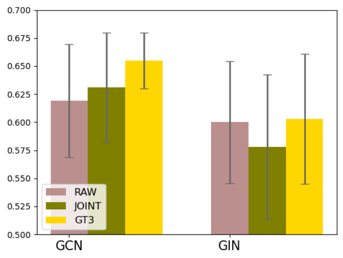
Since $\frac{\eta}{\eta^*} > 0$, we have

$$\mathcal{L}_m(x, y; \boldsymbol{\theta}) - \mathcal{L}_m(x, y; \boldsymbol{\theta}(x)) > 0. \tag{A.19}$$

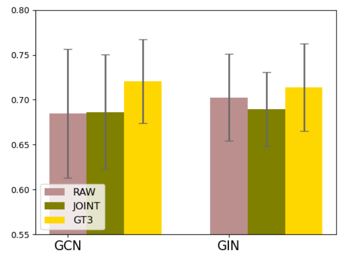
A.2 Performance Comparison on Random Data Split



(a) DD.



(b) ENZYMES.



(c) PROTEINS.

Figure A.1: Performance Comparison of GCN and GIN on Three Datasets on the Random Data Split. Note that the performance metric for ogbg-molhiv is **ROC-AUC**(%) and that for others is **Accuracy**(%).

In addition to the performance comparison on the *OOD* data split, we have also conducted a performance comparison on a random data split, where we randomly split the dataset into

80%/10%/10% for the training/validation/test sets, respectively. The results are shown in Figure A.1. From it, we can observe that GT3 can also improve the classification performance in most cases, which demonstrates the effectiveness of the proposed GT3 in multiple scenarios.

APPENDIX B

CUSTOMIZED GRAPH NEURAL NETWORKS

B.1 Datasets

Some key statistics of seven datasets used in our experiments are shown in Table B.1, and more details of them are introduced as follows:

- **COLLAB** [122] is a dataset of scientific collaboration networks, which describes collaboration pattern of researchers from three different research fields.
- **ENZ** [95] is a dataset of protein tertiary structures of six classes of enzymes.
- **PROT** [9] is a dataset of protein structures, where each graph represents a protein and each node represents a secondary structure element (SSE) in the protein.
- **D&D** [16] is a dataset of protein structures. Each protein is represented as a graph, where each node in a graph represents an amino aid and each edge between two nodes denotes that they are less than 6 Ångstroms apart.
- **RE-BI** and **RE-5K** [122] are two datasets of online discussion threads crawled from different subreddits in Reddit, where each node represents an user and each edge between two users represents their interaction.

Table B.1: The statistics of seven datasets. #Graphs denotes the number of graphs. #Class is the number of graph classes.Node size indicates range, average and standard deviation of the number of nodes among the graphs.Edge size represents range, average and standard deviation of the number of edges among the graphs.

Dataset	#Graphs	#Class	Node size			Edge size		
			range	mean	std	range	mean	std
COLLAB	5000	3	[32,492]	74.5	62.3	[60,40120]	2457.8	6439.0
ENZ	600	6	[2,125]	32.6	14.9	[1,149]	62.14	25.5
PROT	1113	2	[4,620]	39.1	45.7	[5,1049]	72.82	84.6
DD	1178	2	[30,5748]	284.3	272.0	[63,14267]	715.65	693.9
RE-BI	2000	2	[6,3,782]	429.6	554.0	[4,4071]	497.8	623.0
RE-5K	4999	5	[22,3648]	508.5	452.6	[21,4783]	594.9	566.8
NCI109	4127	2	[4,111]	29.6	13.6	[3,119]	32.1	15.0

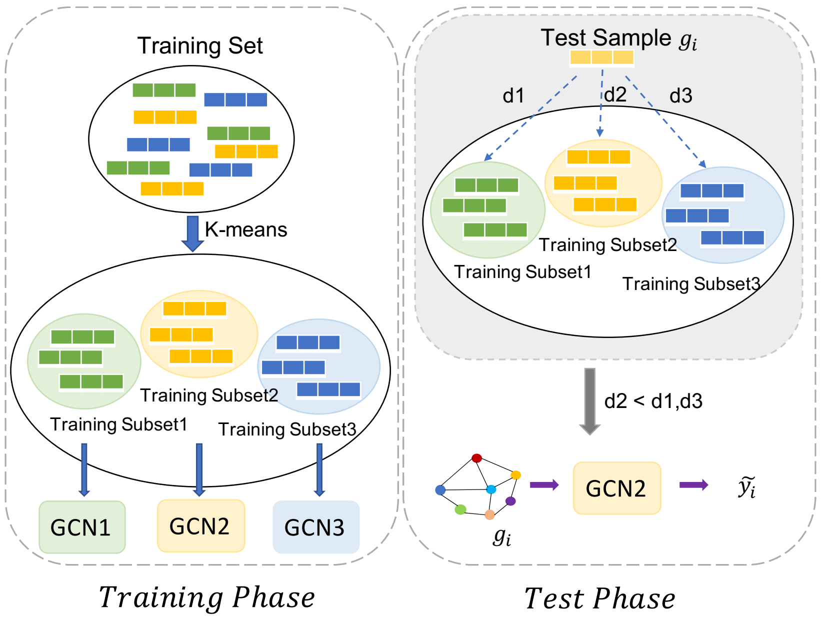


Figure B.1: The framework of Multi-GCN with 3 clusters. we group training samples into 3 clusters and train a GCN model for each cluster. Given a test sample g_i, we measure the distance between this sample and the centroids of the three clusters and utilize the corresponding model of the closest cluster to perform the prediction for this sample.

- **NCI109** [95] is a dataset of chemical compounds screened for activity against non-small cell lung cancer and ovarian cancer cell lines, which are provided by Natinal Cancer Institue (NCI).

B.2 Baselines

In the experiment section, we apply the proposed framework to two graph neural networks: a basic graph convolutional network (GCN) [52] and two SOTA graph classification models DiffPool [126] and gPool [22], respectively. The corresponding adapted versions are Customized-GCN, Customized-DiffPool, and Customized-gPool, respectively. Besides, we also develop baseline methods, Concat-GCN, Concat-Diff, Concat-gPool, Multi-GCN, Multi-Diff, and Multi-gPool. More details of the baseline methods are as follows:

- **GCN** [52] is originally proposed for semi-supervised node classification. It consists of a stack of GCN layers, where a new representation of each node is computed via transforming and aggregating node representations of its neighboring nodes. Finally, a graph representation is generated from node representations in the last GCN layer via a global max-pooling layer and then used for graph classification.

- **Diffpool** [126] is a recently proposed method which has achieved state-of-the-art performance on the graph classification task. It proposes a differentiable graph pooling approach to hierarchically generate a graph-level representation by coarsening the input graph level by level.

- **gpool** [22] is a newly proposed pooling method that has achieved state-of-the-art performance on the graph classification task. It develops a U-Net-like architecture for graph data, consisting of graph pooling operation and unpooling operation based on node importance value.

- **Concat-GCN (or Concat-Diff, Concat-gpool)** is the baseline method that directly concatenates the graph structure properties $\mathbf{s}_i$ to the output graph embedding of the GCN, DiffPool, and gPool model.

- **Multi-GCN (or Multi-Diff, Multi-gpool)** consists of several GCN (or Diffpool, gpool) models trained from different subsets of the training dataset. As shown in Figure B.1, we first cluster data samples from the training set into different training subsets via the K-means method based on the graph structural information. Note that in this work, the structural information $\mathbf{s}_i$ of g_i includes the number of nodes, the number of edges, and the graph density. Then we train different models from different training subsets. During the test phase, given a test graph sample, we first compute the euclidean distance between its graph structural properties and the centroids of different training subsets. Then, we choose the model trained on the closest training subset to do label prediction for this graph sample. In this experiment, we set the number of clusters to 2 and 3, and denote the corresponding frameworks as Multi-GCN-2 (or Multi-Diff-2, Multi-gpool-2) and Multi-GCN-3 (or Multi-Diff-3, Multi-gpool-3). Note that we can also ensemble models trained on all training subsets; however, we empirically find that this strategy does not work.

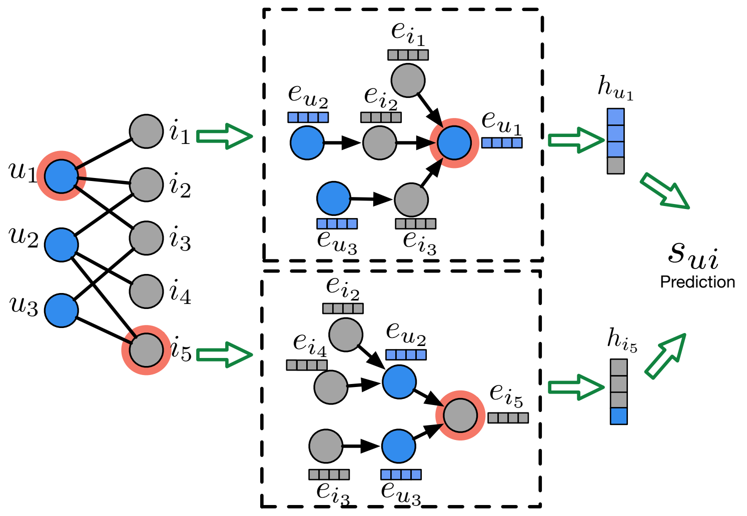
APPENDIX C

LOCALIZED GRAPH COLLABORATIVE FILTERING

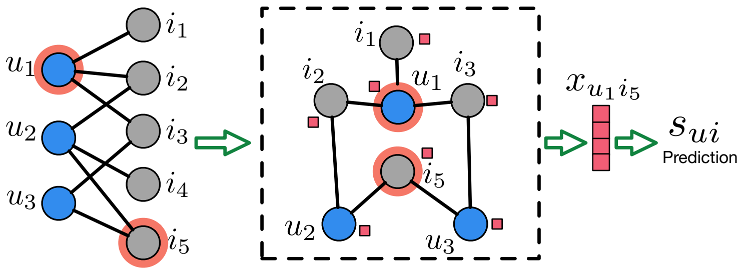
C.1 Further comparison and discussion over two pipelines of GNN-based CF models

The inferior performance of GNN-based methods with sparse interactions may be due to the following two reasons. Firstly, as shown in Figure C.1(a), GNN-based CF methods learn a unique embedding vector e for each user/item from all the historical interactions, which depicts the characteristics of user/item from a global view. The quality of learned embeddings is essential for the desirable recommendations. The user-item interactions denote the supervision information LightGCN and NGCF leverage to learn these user and item representations. Such supervision information in the sparse user-item bipartite graph is not sufficient to learn user and item embeddings of high quality, leading to undesirable recommendation performance. Secondly, the stacked GNN layers contribute to leveraging the high-order information by aggregating neighbors' representations as shown in the middle part of Figure C.1(a). However, given limited interactions, the noises in the learned inferior embeddings may accumulate or even reinforce themselves in this stacked aggregation procedure, which could further degrade the performance [66].

However, the user-item interaction graphs are often sparse in real-world recommendation scenarios [3]. Thus, in this paper, we study how to alleviate the challenge of sparsity while enjoying the merits of high-order topological connectivity. Note that in this work, we focus on addressing the recommendation tasks solely based on the historical interactions following the previous works [110, 35]. Our motivation lies in learning CF signals from local structures of the user-item interaction graph, and the proposed model is called Localized Graph Collaborative Filtering (LGCF). Particularly, given a user and an item, LGCF makes predictions about their interaction based on their local structural context in the bipartite graph, rather than the user and item embeddings. Traditional user and item embeddings are eliminated as LGCF only needs to describe the position of a node relative to the target user and item nodes in the localized graph, which is denoted as red rectangles in Figure C.1(b). Without these intractable user and item embeddings,



(a) Pipeline of GNN-based CF Methods.



(b) Pipeline of the Localized Graph-based CF Model.

Figure C.1: GNN-based CF Methods vs. the Proposed Localized Graph-based CF Model.

LGCF can significantly reduce the number of trainable parameters and thus adapt to the sparse scenario. In addition, the critical CF information is also apparently preserved in the localized graph. The key CF information, i.e., the historical interactions between users and items, are denoted as edges. Therefore, a localized structure covering most edges related to a user-item pair should be able to provide sufficient information to make predictions for this user-item pair. In other words, recommendation-related patterns can be encoded by the localized structures for user-item pairs which motivate LGCF to make recommendations by extracting these patterns. The range of localized structures can be flexibly adjusted to incorporate proper high-order information. In fact, LGCF provides a new perspective to build GNN-based CF models for recommendations where we do not learn a general representation for each user or item but we target on learning a good model to extract the recommendation-related patterns encoded by localized structures of a user-item pair.

Table C.1: Model Size Analysis.

	Model Size		Model Size
MF	$(n+m)d_0$	**NGCF**	$(n+m)d_0 + 2Ld^2$
LightGCN	$(n+m)d_0$	**LGCF**	$Ld^2 + d_L$

C.2 Mode Size Analysis

In this subsection, we discuss and compare the model sizes of the proposed LGCF and existing representative CF methods. In this analysis, we assume that: 1) there are L layers in the GNN model for the GNN-based CF methods, and all layers have the same dimension d; 2) the dimension of user/item embedding for the embedding-based CF methods is d_0, and 3) there are n users and m items in the input user-item bipartite graph. For MF, it only needs to learn user/item embeddings. Thus, its model size is the embedding size, i.e., $(n+m)d_0$. For NGCF, it needs to learn both the user/item embeddings and a GNN model, and thus its model size is the sum of the embedding size and the GNN model size. Note that there are two GNN models in NGCF. One is for user embedding propagation and the other is for item embedding propagation. The weight matrix size in the l-th layer is d^2, thus the GNN model size in total is $2Ld^2$. For LightGCN, it simplifies the GNN model by eliminating feature transformation, therefore its model size is the same as MF. For the proposed LGCF, it does not need to learn embeddings for users and items, thus, its model size is only the sum of a GNN model size and a scoring function size. Note that we adopt a single-layer linear layer for the scoring function, so its model size is d_L, where d_L denotes the output dimension of the GNN model and also the input dimension of the scoring function. The model sizes of LGCF, MF, NGCF and LightGCN are summarized in Table C.1. As shown in the table, the model sizes of baselines are all proportional to the graph size $(n+m)$, which indicates that the models would be very large when the input graph is large. However, the model size of LGCF is independent of the graph size.

C.3 Datasets

We conduct extensive experiments on recommendation datasets from three scenarios. The details of them are introduced as the follows:

- **Tianchi**: This is a user behavior dataset provided by Alibaba for recommendations. There are millions of user-item interactions recorded in this dataset including clicking, purchasing and so on. Each interaction consists of a user ID, an item ID, an item category ID, a behavior type and a timestamp. Note that in our experiments, if there exists a behavior between a user and an item, we consider it as an interaction between them. Specifically, we extract seven user-item interaction datasets from different item categories and denote each of them as Tianchi-ID, where ID indicates the item category. It is common to extract a user-item interaction dataset based on the item category from a large recommendation dataset that has been widely adopted by existing works [71, 110, 35]

- **Amazon**: This is a product review dataset from Amazon, which includes reviews (ratings or texts), product information (e.g., descriptions, category and so on), and user-item interactions (i.e., viewing or purchasing). We select user-item interactions to construct two datasets for items from two categories in Amazon, i.e., Amazon-beauty and Amazon-Gift.
- **MovieLens**: This dataset describes 5-star ratings from *MovieLens*, a movie recommendation service. Each rating record consists of a user ID, a movie ID, a rating score and the category of the movie. In our experiments, we consider each rate as one interaction, regardless of its score.

Specifically, we construct seven datasets corresponding to seven-item categories from Tianchi and denote each of them as Tianchi-ID, where ID indicates the item category, two datasets from two item categories from Amazon, and two datasets from two item categories from MovieLens. The statistics of the datasets are summarized in Table C.2.

C.4 Baselines and Evaluation Metrics

In the experimental parts, we compare the proposed LGCF with MF [53], NGCF [110] and LightGCN [35]. More details of the baseline methods are as follows:

- **MF**: MF aims at directly learning the latent user and item embeddings via minimizing Bayesian Personalized Ranking (BPR) loss. The recommendation prediction is made based

Table C.2: The Statistics of the Datasets from Tianchi, Amazon and MovieLens, respectively.

Dataset	#Users	#Items	#Edges
Tianchi-685988	4,151	539	7,492
Tianchi-2798696	2,579	94	2,988
Tianchi-4527720	1,853	137	2,860
Tianchi-174490	2,267	285	3,826
Tianchi-61626	2,305	764	3,270
Tianchi-810632	5,057	555	9,402
Tianchi-3937919	1,846	158	2,980
Amazon-Beauty	991	85	4,092
Amazon-Gift	458	148	2,966
MovieLens-War	2,970	89	4,240
MovieLens-Romance	20,407	376	33,884

on the inner product of the user embedding and the item embedding.

- **NGCF**: NGCF explicitly encodes the multi-hop connectivity into the embedding learning process. It refines the user and item embeddings via propagation layers, and then makes recommendation predictions based on the refined user embedding and the refined item embedding.

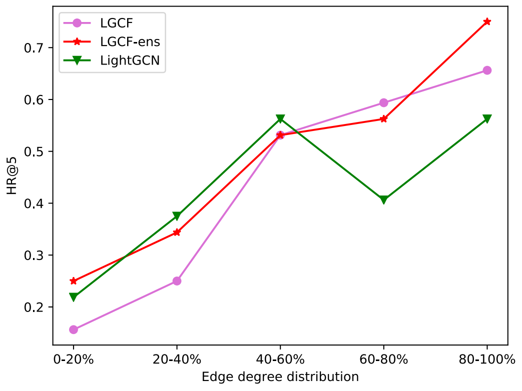
- **LightGCN**: LightGCN is a GNN-based CF method that is customized for recommendations. Specifically, it only includes neighborhood aggregation in GNNs for collaborative filtering, while discarding other components such as feature transformation.

We randomly split the interactions in a dataset into three sets: the training set, the validation set and the test set. To avoid the cold-start setting, we further constrain that all users and items in the validation set and the test set should exist in the training set. More details about the data split will be introduced later. For each user-item interaction in the validation set or the test set, we generate 49 non-existent user-item interactions which share the same user with this user-item interaction. In other words, for the user in a user-item interaction, there are 50 candidate user-item pairs, where only one is positive and the others are negative. During the model evaluation, we compute recommendation scores for 50 candidate user-item pairs for each user-item pair. Then, we rank these scores and calculate the Hit Rate (HR) of the positive pair in the top 5 among 50 candidates and its Normalized Discounted Cumulative Gain (NDCG). Finally, we average HR and

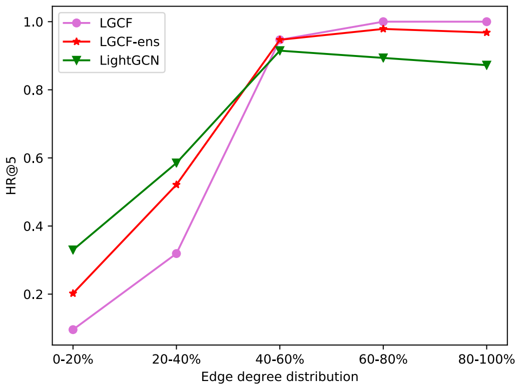
NDCG for all user-item interactions in the validation set or the test set. Specially, we evaluate every model on each dataset with 10 seeds and then report the average performance. In this work, we only report the HR performance since NDCG performance is similar and consistent with HR performance.

C.5 How is LGCF complementary to LightGCN

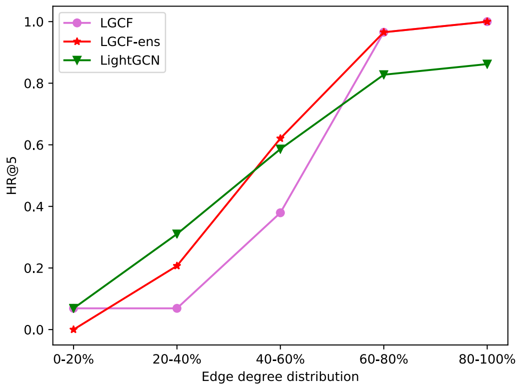
To explore what kind of user-item pairs can benefit from integrating LGCF and LightGCN, we divide the test set in normal scenarios into different groups based on the average degree of the user and the item for a user-item pair. Specifically, we first sort all the user-item pairs in the test set in ascending order according to their average degree and divide the sorted pairs into 5 groups. We then train models on the same training set and test them on these groups. The performance of LightGCN, LGCF, and LGCF-ens on six different datasets is shown in Figure C.2. It can be observed that the performance of all methods tends to increase when the degree increases. In most cases, for all degree groups, LGCF-ens (or integrating LGCF and LightGCN) can boost the recommendation performance.



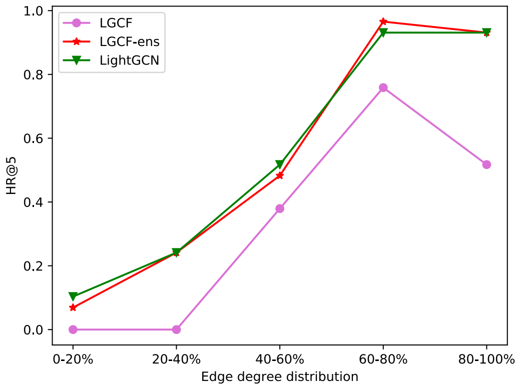
(a) Tianchi-61626.



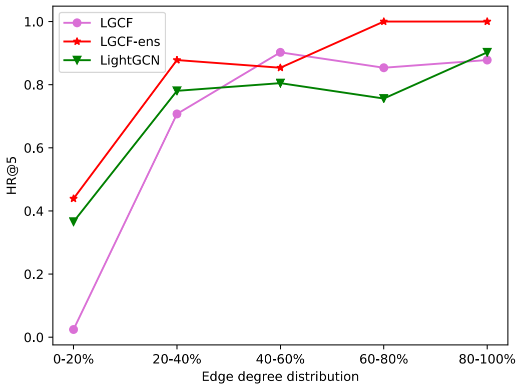
(b) Tianchi-810632.



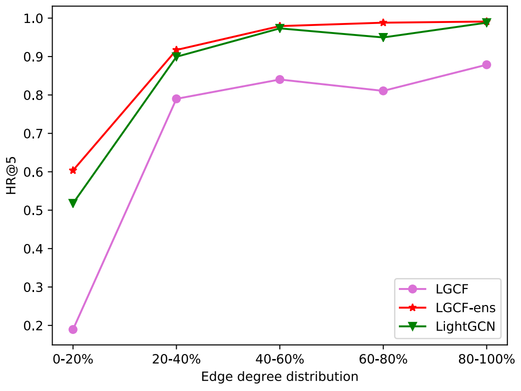
(c) Tianchi-3937919.



(d) Amazon-Gift.



(e) MovieLens-War.



(f) MovieLens-Romance.

Figure C.2: Performance Analysis with Degree.

APPENDIX D

AN ADAPTIVE GRAPH PRE-TRAINING FRAMEWORK FOR LOCALIZED COLLABORATIVE FILTERING

D.1 Datasets

We have summarized the graph property statistics of the 60 pre-training graphs in Table D.1.

D.2 Baselines

The details of these baselines are presented as follows:

- **NGCF**: NGCF [110] proposes to encode high-order connectivity among user-item interaction graphs into user/item embeddings. Specifically, it designs propagation layers to aggregate information from connected nodes. The recommendation prediction is made based on the refined user and item embeddings.

- **LightGCN**: LightGCN [35] is one of the state-of-the-art GNN-based recommendation methods. It tailors GNN particularly for recommendation tasks. Specifically, it eliminates feature transformation and nonlinear activation, and only includes neighborhood aggregation in GNNs for collaborative filtering.

- **MF**: MF [53] is one of the most classic and popular recommendation methods. It learns a set of latent user and item embeddings directly from user-item interactions, and then makes recommendation predictions based on the inner product of the user embedding and the item embedding.

- **GCC**: GCC [84] is a self-supervised pre-training framework for GNNs. GCC designs a subgraph instance discrimination task as the pre-training task and it utilizes contrastive learning to empower GNNs to learn transferable structural representations across multiple graphs. Note that GCC is not specially designed for recommendation tasks. To apply GCC into the recommendation scenarios, we first pre-train a GCN model via the GCC framework on

Table D.1: Graph property statistics of the 60 pre-training graphs from Tianchi.

graph id	the number of nodes	the number of edges	the user-item ratio	the graph density	the degree assortativity coefficient	robins-alexander clustering coefficient	the number of connected components	the global efficiency
1	1066	1426	2.405750799	0.006050346	-0.137080382	0.07622926	58	0.155483363
2	583	961	5.477777778	0.021658778	-0.324017825	0.088734101	15	0.25966763
3	869	1311	4.570512821	0.0117866	-0.229564535	0.075031464	29	0.255150568
4	976	1483	5.731034483	0.012307565	-0.395316067	0.0805384	42	0.21907052
5	825	996	1.272727273	0.005938965	-0.221096689	0.081102362	67	0.116534298
6	547	723	3.240310078	0.013408256	-0.240459448	0.044851669	14	0.204873494
7	442	871	3.20952381	0.02461495	-0.438782979	0.134212489	8	0.28049023
8	879	1606	4.598726115	0.014168005	-0.28871791	0.108867024	17	0.261053578
9	688	1068	3.845070423	0.013774957	-0.391439392	0.08856928	15	0.252946534
10	820	1046	6.192982456	0.012996372	-0.401747324	0.072571549	24	0.189789498
11	1613	2441	7.489473684	0.009028369	-0.364616349	0.060301114	28	0.246112989
12	1828	4100	4.69470405	0.008475505	-0.263666513	0.115984126	21	0.259046914
13	2178	3621	11.88757396	0.010665025	-0.468666839	0.117659122	10	0.233061438
14	2182	3784	7.693227092	0.007807197	-0.246347149	0.070869752	33	0.241451953
15	1455	2201	3.575471698	0.006087409	-0.203866274	0.115162664	49	0.165451187
16	3112	4364	9.103896104	0.005053078	-0.271381784	0.045556577	45	0.237112946
17	1791	2589	10.05555556	0.009810609	-0.423324113	0.062591444	15	0.247061121
18	1632	2319	7.201005025	0.008132077	-0.292355843	0.08403141	45	0.19445713
19	2133	2901	1.83643617	0.00279342	-0.173274517	0.060554047	122	0.133096639
20	2191	3995	7.834677419	0.008290721	-0.378478248	0.091360009	17	0.257314823
21	2986	4045	9.514084507	0.005271265	-0.334648858	0.018636375	16	0.349629458
22	1102	2003	5.482352941	0.01264201	-0.391299794	0.088862343	11	0.249883801
23	1667	2660	10.34013605	0.011904762	-0.262163651	0.057379615	8	0.282671889
24	1264	1858	6.479289941	0.010040258	-0.401666782	0.056050426	17	0.231782658
25	1224	2145	3.25	0.007957176	-0.236265509	0.071828792	38	0.212799377
26	1105	1858	4.846560847	0.010732192	-0.278355787	0.081693911	15	0.247897105
27	1631	3520	18.65060241	0.027396407	-0.528782764	0.140792463	6	0.284775151
28	1344	1786	6.466666667	0.008524246	-0.295103378	0.267301969	31	0.196343865
29	2400	3008	7.391608392	0.004975157	-0.249780826	0.076222519	64	0.140260529
30	2231	3660	5.639880952	0.00574821	-0.360584068	0.076936354	61	0.208970493
31	2432	4992	11.28282828	0.011285641	-0.358731221	0.099939883	24	0.276760795
32	4669	7776	4.182019978	0.002290449	-0.272368625	0.052135774	82	0.203264965
33	4144	5838	5.6304	0.00265439	-0.231825096	0.073917706	161	0.161605432
34	4224	6530	5.148471616	0.002687332	-0.180418703	0.046033635	112	0.188019456
35	4461	7107	7.764243615	0.003533065	-0.20104595	0.050613051	70	0.219329989
36	4380	6090	8.106029106	0.003247274	-0.239028273	0.062308213	117	0.20649004
37	3745	7738	3.734513274	0.00331163	-0.245603955	0.063679839	75	0.235388263
38	2168	4729	3.6825054	0.005990512	-0.238712029	0.061400362	26	0.235880829
39	4123	5174	21.90555556	0.007289994	-0.27692024	0.059894654	40	0.219759568
40	2930	4631	2.123667377	0.002478464	-0.159829692	0.03445955	130	0.164731802
41	4082	6653	3.38453276	0.002267877	-0.249515119	0.047776691	116	0.177176302
42	2406	4571	2.51754386	0.003880806	-0.245047333	0.046862868	63	0.19723984
43	2659	7717	5.581683168	0.008470725	-0.306202709	0.074841219	10	0.27057376
44	4892	7366	7.814414414	0.003060196	-0.184142643	0.040597988	30	0.251175191
45	3372	6459	5.386363636	0.00430132	-0.307159357	0.047727729	32	0.236428669
46	2413	5228	3.339928058	0.005063477	-0.311985584	0.065333207	46	0.208494542
47	3130	5607	5.804347826	0.004565217	-0.288644888	0.050126535	54	0.220249111
48	3921	5942	1.584706658	0.001629343	-0.181370456	0.026907729	161	0.160805618
49	2654	4640	2.005662514	0.002967145	-0.202301709	0.070429402	73	0.183938794
50	3379	5260	4.287949922	0.003004238	-0.271414349	0.05995576	142	0.182738023
51	5401	9717	6.060130719	0.002739853	-0.27968436	0.044058991	117	0.197594675
52	5364	9430	15.05988024	0.005613028	-0.248281597	0.056374776	29	0.276165783
53	6849	10503	3.97747093	0.001394664	-0.132708191	0.058927812	195	0.16746493
54	6479	10583	10.90992647	0.003277851	-0.253056336	0.049755793	76	0.229149938
55	5865	8956	5.495016611	0.001998801	-0.251292838	0.060734676	167	0.174011088
56	12284	19924	5.478902954	0.001011594	-0.259199682	0.038720139	332	0.16613786
57	8056	18171	8.154545455	0.002877489	-0.282159789	0.060062896	49	0.250108737
58	7927	17428	17.05694761	0.005301725	-0.366123756	0.102245138	22	0.280886077
59	6970	19142	4.813177648	0.002766413	-0.290319073	0.046690635	72	0.237782573
60	7131	14532	7.869402985	0.002856745	-0.210269642	0.063454378	101	0.226749207

the pre-training graphs. Then we build a scoring function model specific to recommendations in the fine-tuning phase. The fine-tuning objective function is the same as ADAPT.

- **InfoGraph**: InfoGraph [99] is an unsupervised graph learning method based on contrastive learning. It trains the GNNs via maximizing the mutual information between the local-level representation and graph-level representation. To apply InfoGraph into the proposed scenario, we pre-train a GCN model via InfoGraph on the pre-training graphs, and then fine-tune the GCN model and train a recommendation scoring function model on the downstream graph.
- **MVGRL**: MVGRL [31] is also an unsupervised graph learning method based on contrastive learning. It contrasts the node-level representation and a graph-level representation from a graph diffusion view and a regular view. To apply MVGRL into the proposed scenario, we first pre-train a GCN model following MVGRL on the pre-training graphs, and then do fine-tuning on the downstream graph and also train a recommendation scoring function model.

D.3 Implementation Details

In our work, we implement the meta-LGNN model as a 3-layer graph convolutional networks and the scoring function as a linear transformation layer. For the input of the GNN adaptor, we compute eight graph properties via the networkX package [26], including the number of nodes, the number of edges, the user-item ratio, the graph density, the degree assortativity coefficient, robins-alexander clustering coefficient, the number of connected components and the global efficiency. Note that it is straightforward to include more graph properties. We use the Adam [50] optimizer. The batch size is set to 256. The learning rate is chosen from $[0.0005, 0.001]$, and the dropout rate is chosen from $[0.0, 0.1, 0.2, 0.3]$.