UTS UNIVERSITY OF TECHNOLOGY SYDNEY

Deep Learning on Abnormal and Evolving Graphs

by Chaoxi Niu

Thesis submitted in fulfilment of the requirements for the degree of

Doctor of Philosophy

under the supervision of Ling Chen and Yulei Sui

University of Technology Sydney
Faculty of Engineering and Information Technology

July 2025

CERTIFICATE OF ORIGINAL AUTHORSHIP

I, Chaoxi Niu, declare that this thesis is submitted in fulfilment of the requirements for the award of Doctor of Philosophy, in the Faculty of Engineering and Information Technology at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution.

This research was supported by an Australian Government Research Training Program (RTP) Scholarship doi.org/10.82133/C42F-K220.

Signature: Production Note: Signature removed prior to publication.

Date: 14/07/2025

ABSTRACT

Deep Learning on Abnormal and Evolving Graphs

by

Chaoxi Niu

Graphs are widely used to model complex relationships between data instances. Graph Neural Networks (GNNs), which integrate node attributes and graph structures, have emerged as a powerful tool for deep graph learning. However, most existing GNNs focus on regular graphs that are clean and static, limiting their effectiveness in real-world scenarios where graphs are often irregular, *i.e.*, graphs are either contaminated by anomalies or evolving over time. This thesis addresses the underexplored yet practically important abnormal and evolving graph learning. Specifically, abnormal graph learning aims to detect and reject anomalous nodes or graphs that deviate from expected patterns, which is known as graph anomaly detection. In contrast, evolving graph learning aims to enable GNNs to learn from sequentially arriving graph tasks without forgetting previously seen ones. The exploration of this thesis is driven by the following issues that hinder the effectiveness of GNNs for abnormal and evolving graph learning.

For graph anomaly detection, existing works mostly focus on detecting abnormal nodes or edges in a single graph, leading to graph-level anomaly detection being significantly underexplored. Besides, existing detection methods are one-model-for-one-dataset approaches, *i.e.*, training a separate model for each graph dataset. This largely limits their applicability in real-world scenarios. For evolving graph learning, memory replay-based methods, which aim to replay historical data when learning new tasks, have been explored as one principled approach. However, the constructed memory fails to capture the holistic graph information and raises privacy concerns. Moreover, among the two settings of evolving graph learning, the absence of task

identifiers significantly degrades the performance of class-incremental learning than that of task incremental learning.

To tackle these problems, four models are proposed in this thesis to enhance GNNs for abnormal and evolving graphs. Firstly, a hierarchical memory network is designed, which learns hierarchical memory modules via a graph autoencoder network architecture for graph-level anomaly detection. Secondly, a novel zero-shot generalist graph anomaly detection approach is proposed that trains a one-for-all model, requiring the training of one detection model on a single graph dataset and then effectively generalizing to detect anomalies in other graph datasets without any retraining or fine-tuning. Then, to capture the holistic information for memory replay, a debiased lossless memory replay method is proposed. Lastly, to bridge the performance gap between task- and class-incremental learning, a replay-and-forget-free method is proposed with Laplacian smoothing-based graph task profiling and graph prompting. Extensive experiments across diverse graph datasets demonstrate the effectiveness of the proposed models, which achieve state-of-the-art performance and offer new insights into deep learning on irregular graphs.

Dissertation directed by Professor Ling Chen

AAII - Australian Artificial Intelligence Institute, UTS

Dedication

To my family.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Prof. Ling Chen, for granting me the invaluable opportunity to pursue a Ph.D. under her guidance. Throughout every stage of my doctoral journey, she has provided continuous support, insightful feedback, and patient mentorship. Her sincere attitude to research and life has been a constant source of inspiration, reminding me to focus on what truly matters.

I am also sincerely thankful to my external supervisor, Prof. Guansong Pang, for his invaluable mentorship and consistent guidance. I am especially grateful for his consistent weekly supervision and hands-on support across all my research projects. His dedication, high standards, and insightful feedback have significantly shaped my academic development.

I am profoundly grateful to my family and close relatives, especially my mother, for their unconditional support during the most challenging times. My heartfelt thanks go to my partner, Changlu Chen, for her unwavering love and encouragement, which have been a constant source of strength. Lastly, I thank all the friends and colleagues I have met so far for their kindness and companionship, which made this experience more meaningful and joyful.

Chaoxi Niu

Sydney, Australia, 14/07/2025.

List of Publications

1. **Niu, C.**, Pang, G., Chen, L. (2023). Graph-level anomaly detection via hierarchical memory networks. ECML-PKDD. (Core A)

2. **Niu, C.**, Pang, G., Chen, L. (2024). Affinity uncertainty-based hard negative mining in graph contrastive learning. TNNLS. (Core A*)

3. **Niu, C.**, Pang, G., Chen, L. (2024). Graph continual learning with debiased lossless memory replay. ECAI. (Core A)

4. **Niu, C.**, Pang, G., Chen, L., Liu, B. (2024). Replay-and-forget-free graph class-incremental learning: A task profiling and prompting approach. NeurIPS. (Core A*)

5. **Niu, C.**, Qiao, H., Chen, C., Chen, L., Pang, G. (2025) Zero-shot generalist graph anomaly detection with unified neighborhood prompts. IJCAI. (Core A*)

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Graph plays an important role in a wide range of domains [1–3], such as social networks, traffic networks, biological protein-protein networks, and recommendation systems. Generally, graphs capture interactions (*i.e.*, edges) between individual units (*i.e.*, nodes), providing both the feature and structure information. Following a neighborhood aggregation mechanism to capture both graph structure and node/edge features, graph neural networks (GNNs) have become predominant in learning effective graph representations, which facilitate downstream tasks such as graph classification, node classification, link prediction, and recommendation [4, 5].

Most existing GNNs are designed under the assumption that graphs are clean and static [1, 2, 6]. However, this assumption of regular graphs barely holds in real-world applications, where graphs are often irregular—either corrupted by anomalies [7–9] or evolving over time [10–12]. Such irregularities render standard GNNs ineffective. Motivated by this, this thesis focuses on the more challenging yet practical problems of abnormal and evolving graph learning, which aims to advance graph neural networks to operate robustly on irregular graphs and better reflect the noisy and dynamic nature of real-world data.

Abnormal Graph Learning For abnormal graph learning, it aims to identify the graph anomalies, which is also known as graph anomaly detection [7–9]. Specifically, anomalies in the graph can significantly impact the performance and reliability of

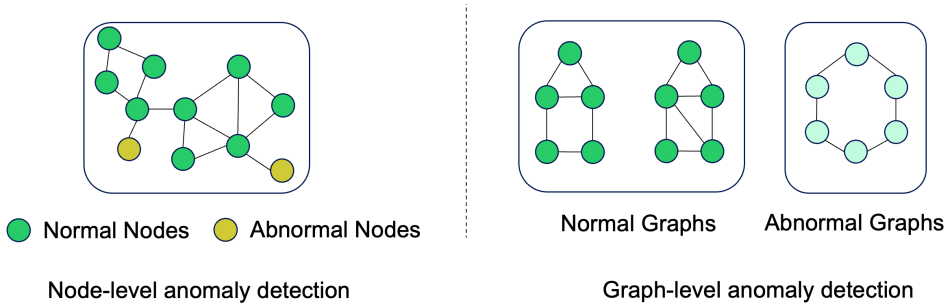


Figure 1.1 : Node-level and graph-level anomaly detection.

graph neural networks. By identifying and addressing such anomalies, graph neural networks can better capture the intrinsic patterns of graph data.

Graph anomaly detection can be roughly divided into two categories: node-level anomaly detection and graph-level anomaly detection, as shown in Figure 1.1. Specifically, node-level anomaly detection aims to identify nodes in a graph that significantly deviate from normal patterns, and graph-level anomaly detection aims to identify abnormal graphs that exhibit deviant structures and node attributes compared to the majority in a graph set. Both of them have wide applications in detecting abnormal instances in a variety of graph/network data, e.g., abusive user behaviors in online user networks, fraudulent activities in financial networks, and spam in social networks. Recently, numerous works have been proposed for graph anomaly detection. However, most of them focus on node-level anomaly detection, resulting in graph-level anomaly detection being less explored. Generally, anomalous graphs can be any graphs that are abnormal in part or in whole, which are referred to as locally anomalous or globally anomalous graphs. Therefore, to effectively detect anomalous graphs, the normal patterns within both fine-grained and holistic views of graphs must be captured. In addition to the less-explored graph-level anomaly detection, one noticeable observation is that existing graph anomaly detection approaches need to train a dataset-specific graph anomaly detection model for each

graph dataset. This one-model-for-one-dataset paradigm limits their applicability in real-world scenarios since training a model from scratch can incur extensive computation costs and require a large amount of labeled data for supervised graph anomaly detection methods on each dataset. Inspired by the universal capacity of foundation models in natural language processing and computer vision, it is imperative to learn a generalist detection model on auxiliary datasets so that it can generalize to detect anomalies effectively in diverse target datasets without any retraining or fine-tuning. However, the heterogeneous node attribute distributions and the lack of a generalized anomaly metric across graphs are the main obstacles to achieving this goal.

Evolving Graph Learning Different from abnormal graph learning which aims to identify and reject the abnormal data, evolving graph learning focuses on adapting graph neural networks for new graphs while memorizing previous graph patterns.

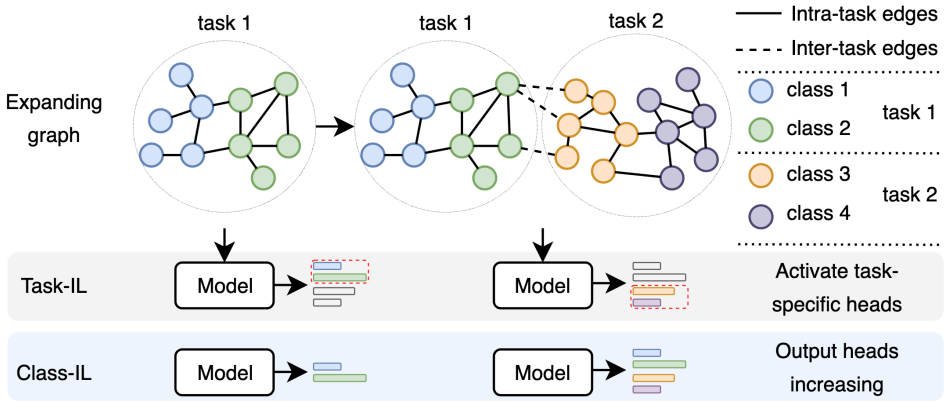


Figure 1.2 : Illustration of task-incremental learning and class-incremental learning.

In real-world applications, graphs often expand continually. For example, in the context of online shopping, consumers could be nodes, and the edges between consumers could represent that they have purchased or rated the same product. New consumers and the associated connections would be constantly added to the

online shopping network. The newly emerged graph is referred to as a new graph learning task. The continually expanding graphs render the learning of graph neural networks on static graph data impractical. This is because directly applying GNNs to accommodate new emerging graphs would cause the GNNs to easily forget the knowledge learned from the previous data due to the large distribution difference between the historical data and the newly added data. The forgetting of the learned knowledge when learning new graphs, a.k.a. catastrophic forgetting, would result in deteriorated performance on historical graphs. A simple solution is to store all historical data and repeatedly retrain the GNNs whenever the graph is updated, but it is prohibitively expensive in terms of computational time and resources. To address these issues for evolving graph learning, graph continual learning [11–13] is proposed to continually adapt GNNs to the expanded graph of the current task while maintaining the performance over the graph of previous tasks.

For graph continual learning, two settings are widely explored, *i.e.*, task-incremental learning and class-incremental learning as shown in Figure 1.2. The difference between them is the absence of task identifiers during inference in class-incremental learning. More specifically, class-incremental learning requires classifying the test instance into one of all the classes of all the learned tasks, while task-incremental learning only needs to distinguish test instances between classes within a known task. Existing graph continual learning methods can be roughly divided into three categories: regularization-based methods, parameter isolation-based methods, and replay-based methods. Due to the straightforward intuitiveness and impressive effectiveness, replay-based methods have been widely explored and have achieved remarkable capacity against catastrophic forgetting. However, existing replay-based methods struggle to capture holistic information for memory construction, leading to degraded performance. Furthermore, as discussed before, class-incremental learning does not have access to task identifiers during inference. This presents an additional

challenge for graph continual learning, known as inter-task class separation [14–16], *i.e.*, the class separation in one graph task is obstructed by the presence of classes from other tasks. As a result, the performance of class-incremental learning is significantly far below that of task-incremental learning. To bridge the performance gap, a feasible approach is to predict the task identifier of test instances in class-incremental learning. There have been some studies on task prediction in continual learning [14–16], but they are designed for Euclidean data and cannot be directly used for graph continual learning.

1.2 Research Objectives

This thesis focuses on deep learning on abnormal and evolving graphs. Abnormal graph learning aims to detect and reject anomalous nodes or graphs that deviate from expected patterns. Evolving graph learning aims to enable GNNs to learn from sequentially arriving graph tasks without forgetting previously seen ones. Specifically, the exploration of this thesis is guided by the following research objectives.

- Firstly, compared to node-level anomaly detection, graph-level anomaly detection is significantly less explored, despite its great importance and broad application. Graph-level anomaly detection aims to identify abnormal graphs that exhibit deviant structures and node attributes compared to the majority in a graph set. One primary challenge is to learn normal patterns manifested in both fine-grained and holistic views of graphs for identifying graphs that are abnormal in part or in whole.

  To tackle this challenge, this thesis proposes a novel approach called Hierarchical Memory Networks (HimNet), which learns hierarchical memory modules – node and graph memory modules – via a graph autoencoder network architecture. The node-level memory module is trained to model fine-grained, internal

graph interactions among nodes for detecting locally abnormal graphs, while the graph-level memory module is dedicated to the learning of holistic normal patterns for detecting globally abnormal graphs. The two modules are jointly optimized to detect both locally- and globally-anomalous graphs.

- Secondly, existing graph anomaly detection methods are one-model-for-one-dataset approaches, *i.e.*, training a separate model for each graph dataset. This largely limits their applicability in real-world scenarios.

  To overcome this limitation, this thesis proposes a novel zero-shot generalist graph anomaly detection approach, UNPrompt, that trains a one-for-all detection model, requiring the training of one model on a single graph dataset and then effectively generalizing to detect anomalies in other graph datasets without any retraining or fine-tuning. The key insight is that i) the predictability of latent node attributes can serve as a generalized anomaly measure and ii) generalized normal and abnormal graph patterns can be learned via latent node attribute prediction in a properly normalized node attribute space. UNPrompt achieves a generalist mode for graph anomaly detection through two main modules: one module aligns the dimensionality and semantics of node attributes across different graphs via coordinate-wise normalization, while another module learns generalized neighborhood prompts that support the use of latent node attribute predictability as an anomaly score across different datasets.

- Thirdly, for graph continual learning, memory replay-based methods, which replay data of previous tasks when learning new tasks, have been explored as one principled approach to mitigate the forgetting of the knowledge learned from the previous tasks. However, the sampling-based memory of these methods struggles to capture the holistic information and raises privacy concerns.

To address this problem, this thesis designs a method to learn prototypical node representations as memory. The learned memory can not only preserve the graph data privacy but also capture the holistic graph information, both of which the sampling-based methods fail to achieve. Further, prior methods suffer from bias toward the current task due to the data imbalance between the classes in the memory data and the current data. A debiased memory-replay loss function is also devised to alleviate this bias.

- Lastly, graph class-incremental learning aims to continually learn a sequence of tasks, with each task consisting of a set of unique classes. The key characteristic of class-incremental learning lies in the absence of task identifiers during inference, which causes a significant challenge in separating classes from different tasks. Being able to accurately predict the task identifiers can help address this issue, but it is a challenging problem.

  To tackle this challenge, this thesis shows theoretically that accurate task identifier prediction on graph data can be achieved by a Laplacian smoothing-based task profiling approach, in which each graph task is modeled by a task prototype. It guarantees that the task prototypes of the same graph task are nearly the same with a large smoothing step, while those of different tasks are distinct due to differences in graph structure and node attributes. Further, to avoid the catastrophic forgetting of the knowledge learned in previous graph tasks, a novel graph prompting approach is proposed, which learns a small discriminative graph prompt for each task, essentially resulting in a separate classification model for each task.

1.3 Contribution

The contributions of this thesis are summarized as follows:

- A hierarchical node-to-graph memory network is introduced for graph-level anomaly detection, which is the first memory-based work. Specifically, a three-dimensional node memory module consists of multiple two-dimensional memory blocks (with each block capturing one type of normal pattern on the representations of all nodes), and a graph memory module enables each memory block to capture graph-level normal patterns. These two memory modules are learned by jointly minimizing a graph reconstruction error and a graph approximation error.

- A novel zero-shot generalist graph anomaly detection approach is proposed. To the best of our knowledge, this is the first method that exhibits effective zero-shot graph anomaly detection performance across various graph datasets. By unifying the heterogeneous distributions of the node attributes across different graphs, we further introduce a novel neighborhood prompt learning module that utilizes a neighborhood-based latent node attribute prediction task to learn generalized prompts. This enables the zero-shot anomaly detection across different graphs.

- A novel learnable memory replay-based approach is designed. To obtain such memory, a lossless memory learning method is proposed to utilize gradient matching to enforce a lossless compression of large graphs of previous tasks into a small set of prototypical node representations as the memory data. It achieves not only a small memory budget but also good privacy preservation of historical data. To mitigate the bias toward the current graph, we further devise a debiased objective that effectively calibrates the predictions based on label frequencies.

- A novel graph task profiling and prompting approach is proposed, which is the first replay- and forget-free graph class-incremental learning approach. A

simple Laplacian smoothing-based task profiling approach is introduced and can achieve accurate graph task prediction, which eliminates the inter-task class separation issue. To the best of our knowledge, it is the first work that leverages the non-Euclidean properties of graph data to enable graph task prediction. Furthermore, a novel graph prompting approach is proposed to learn a small prompt for each task using a frozen pre-trained GNN. With the support of the accurate task prediction, the graph prompts result in a separate classification model for each task, leading to replay and forget-free graph continual learning.

- Extensive experiments and ablation studies are conducted on various graph datasets, and state-of-the-art performance has been achieved to verify the superiority and effectiveness of each component of the proposed methods.

1.4 Thesis Organization

This thesis is organized as follows:

- *Chapter 2*: This chapter presents a survey on graph anomaly detection and graph continual learning.
- *Chapter 3*: This chapter proposes a hierarchical memory network, which learns hierarchical memory modules to detect locally- and globally-anomalous graphs.
- *Chapter 4*: This chapter presents a generalist zero-shot graph anomaly detection approach to learn a generalist model on an auxiliary dataset that can directly generalize to other unseen graphs.
- *Chapter 5*: This chapter designs a debiased learnable memory-based replay framework to enhance the effectiveness of memory replay for graph continual learning.

- *Chapter 6*: This chapter proposes a task profiling and graph prompting approach for graph class-incremental learning to overcome the problem of missing task identifiers during inference.

- *Chapter 7*: The final chapter provides a summary of the thesis, as well as the potential future works.

Chapter 2

Literature Review

2.1 Deep Learning for Graph

Early approaches for graph learning typically rely on handcrafted structural features, graph kernels, and spectral methods [17–19]. However, these methods often face scalability issues and struggle to capture high-level semantics. Recently, the introduction of Graph Neural Networks (GNNs) [1, 2, 6] has reshaped the field by generalizing deep learning to non-Euclidean domains. Graph Convolutional Networks (GCNs) [20] employ spectral filters and message passing to propagate local information, while Graph Attention Networks (GATs) [21] introduce attention mechanisms for adaptive neighborhood aggregation. GraphSAGE [22] addresses scalability by sampling local neighborhoods, enabling inductive learning. Further extensions have explored deeper architectures [23], higher-order structures [24, 25], and subgraph-level representations [26] to improve the expressiveness of GNNs. Despite the remarkable successes achieved by these GNNs, they primarily focus on regular graphs, which barely hold in real-world applications. Specifically, graphs are often corrupted by anomalies or evolve over time, rendering standard GNNs ineffective. Motivated by this, this thesis focuses on advancing deep learning on abnormal and evolving graphs.

2.2 Graph Anomaly Detection

2.2.1 Node-level Anomaly Detection

Existing graph anomaly detection methods can be roughly categorized into unsupervised and supervised approaches [8, 9]. The unsupervised methods are typically built using data reconstruction, self-supervised learning, and learnable graph anomaly measures [9, 27]. The reconstruction-based approaches like DOMINANT [28] and AnomalyDAE [29] aim to capture the normal patterns in the graph, where the reconstruction error in both graph structure and attributes is utilized as the anomaly score. CoLA [30] and SL-GAD [31] are representative self-supervised learning methods, assuming that normality is reflected in the relationship between the target node and its contextual nodes. The graph anomaly measure methods typically leverage the graph structure-aware anomaly measures to learn intrinsic normal patterns for graph anomaly detection, such as node affinity in TAM [32]. In contrast to the unsupervised approaches, the supervised approaches have shown substantially better detection performance in recent years due to the incorporation of labeled anomaly data [33, 34]. Most supervised methods concentrate on the design of propagation mechanisms and spectral feature transformations to address the notorious over-smoothing issues [34–36].

2.2.2 Graph-level Anomaly Detection

Recently, a few graph-level anomaly detection methods have been proposed. These works can be divided into two categories: two-step methods and end-to-end methods. The first category typically obtains vectorized graph representations using graph kernels (e.g., Weisfeiler-Leman Kernel [17] and propagation kernels [37]), or advanced graph neural networks (such as Graph2Vec [38] and InfoGraph [39]). An off-the-shelf anomaly detector is then applied to the learned graph representations to detect abnormal graphs, such as k-nearest-neighbor distance [40], isolation

forest [41], local outlier factor [42], and one-class support vector machine [43]. However, the two-step methods may achieve suboptimal performance since the anomaly detectors are independent of the graph representation learning. To address this issue, end-to-end methods unify graph representation learning and anomaly detection. Typically, they utilize powerful GNNs as the backbone and learn graph representations tailored for graph anomaly detection. For example, [44] applied the Deep SVDD objective [45] on top of the GNN-based graph representations for anomalous graph detection. [46] utilized random knowledge distillation on both node and graph representations to capture graph regularity information. Some works also employed the contrastive learning strategy for detecting anomalous graphs [47–49]. These methods show better performance than the two-step methods, but they focus on learning discriminative feature representations only, which may fail to preserve the primary graph semantics.

2.2.3 Generalist Anomaly Detection

Generalist anomaly detection has recently emerged as a promising solution to tackle sample efficiency and model generalization problems in anomaly detection. There have been a few studies on non-graph data that have large pre-trained models to support the generalized pattern learning, such as image generalist anomaly detection [50,51]. However, it is a very challenging task for data like graph data due to the lack of such pre-trained models. Recently, a concurrent approach, ARC [52], introduces an effective framework that leverages in-context learning to achieve generalist graph anomaly detection without relying on large pre-trained GNNs. However, ARC focuses on a few-shot graph anomaly detection setting, *i.e.*, requiring the availability of some labeled nodes in the target testing graph dataset, which might be difficult to get in real applications.

2.3 Graph Continual Learning

Graph continual learning has gained growing popularity in deep learning, and various methods have been proposed [53–63], which can be divided into three categories, *i.e.*, regularization-based, parameter isolation-based, and data replay-based methods. Regularization-based methods typically preserve parameters that are important to the previous tasks when learning new tasks via additional regularization terms. For example, TWP [55] preserves the important parameters in the topological aggregation and loss minimization for previous tasks via regularization terms. Parameter isolation-based methods maintain the performance on previous tasks by continually introducing new parameters for new tasks, such as [62] proposes to continually expand model parameters to learn new emerging graph patterns. Differently, replay-based methods [54, 61, 63–65] employ an additional memory buffer to store the information of previous tasks and replay them when learning new tasks. A typical example of the replay-based method is ERGNN [54]. However, the graph structure, which plays a vital role in graph representation learning, is not considered in ERGNN. To cope with the topological information, [61] and [63] proposed sparsification techniques to find the important neighbors of the selected nodes. Then, the important neighbors together with the edges between neighbors and representative nodes are stored in the memory buffer for replaying during the learning of new tasks. Nevertheless, all these methods focus on constructing the memory using partial graph data of previous graphs, failing to preserve the holistic graph semantics and also raising privacy concerns in privacy-critical applications. Moreover, although they have shown good performance in alleviating the forgetting problem, inter-class separation is still a significant challenge to these methods, especially for the class-incremental learning where the task IDs are not provided when testing. To improve the class-incremental learning performance, an emerging research direction focuses on performing task identifiers prediction during testing. For example,

CCG [66] utilizes a separate network for task identification. HyperNet [67] and PR-Ent [68] use the entropy to predict the task of the test sample. More recently, [14] proves that out-of-distribution detection is the key to task prediction and proposes an identification method based on an out-of-distribution detector. TPL [16] further improves it by exploiting the information available in class-incremental learning for better task identification. However, these methods were designed for Euclidean data, resulting in them not being suited for graph class-incremental learning.

Chapter 3

Graph-level Anomaly Detection via Hierarchical Memory Networks

3.1 Introduction

Graphs are widely used to model complex relationships between data instances in various fields, such as social networks, bioinformatics, chemistry, etc. Graph neural networks (GNNs) have become the predominant approach to learning effective node/graph representations and have achieved impressive performance in many graph-related tasks, such as node classification [20], link prediction [69] and graph classification [70]. Despite the remarkable success achieved by GNNs, it is still challenging for GNNs to tackle some notoriously difficult tasks. Graph-level anomaly detection (GLAD), which aims to identify abnormal graphs that exhibit deviant structures and node attributes in comparison to the majority in a set of graphs, is one of such tasks.

In recent years, a number of graph anomaly detection methods have been proposed. However, a majority of them focus on the detection of abnormal nodes or edges in a single graph [30, 71–75]. In contrast, graph-level anomaly detection is significantly less explored, despite its great importance and broad application [18,46,76]. In general, anomalous graphs can be any graphs that are abnormal in part or in whole, which are referred to as locally-anomalous or globally-anomalous graphs [46, 47]. The local abnormality requires a fine-grained inspection of the graphs, as it is primarily due to the presence of unusual local graph structures, e.g., nodes and their associated local neighborhoods, compared to the correspond-

ing structures in the other graphs. The global abnormality, on the other hand, requires a holistic treatment of the graphs, as it is manifested only at the graph-level representations. Thus, the main challenge in GLAD is to learn normal patterns from both fine-grained and holistic views of graphs for identifying both locally- and globally-anomalous graphs.

A few GLAD methods have been introduced, e.g., [46, 47]. They employ knowledge distillation [46] or contrastive learning [47] on the node and graph representations to capture the local/global normal patterns. The key intuition of these methods is that the model trained to fit exclusively normal training graphs learns normality representations, on which abnormal test graphs would be discriminative from the normal graphs. Despite their effectiveness, the learned normality representations may not preserve the primary semantics of graph structures and attributes, since their learning objectives ignore these semantics and focus on enlarging the relative difference between normal and abnormal graphs in the representation space. Consequently, they become ineffective in detecting abnormal graphs in which semantic-rich graph representations are required.

This chapter introduces a novel approach, namely hierarchical memory networks (**HimNet**), via a graph autoencoder network architecture to learn hierarchical node and graph memory modules for GLAD, which not only help effectively differentiate normal and abnormal graphs but also preserve rich primary semantics. Autoencoder (AE) [77, 78], which utilizes a decoder to reconstruct the original input based on the representations learned by an encoder, is a widely-used approach to preserve the rich semantics of the input data in the new representation space. AE is also commonly used for anomaly detection in various domains [79–84] since anomalies are generally difficult to reconstruct, and thus, they have a higher reconstruction error than normal samples. However, reconstructing graphs is difficult since it involves the reconstruction of diverse graph structures and attributes. Our hierarchical memory

learning is designed to address this issue. Specifically, the node-level memory module captures the local normal patterns that describe the fine-grained, internal graph interactions among nodes, and it is optimized by minimizing *a graph reconstruction error* between original input graphs and the graphs reconstructed from the node memory module. On the other hand, the graph-level memory module is dedicated to the learning of holistic normal patterns of graph-level representations, and it is optimized by minimizing *a graph approximation error* between graph-level representations and their approximated representations based on the graph memory. The two modules are jointly optimized to detect both locally and globally anomalous graphs. Memory-augmented AEs [83, 84] have been introduced to add an additional memory module for anomaly detection in image and video data. The memory module has shown promise in regularizing AEs, enabling significantly improved detection performance. However, their memory module is not applicable to graph data in a non-Euclidean space. HimNet addresses this problem by learning hierarchical node and graph memory modules to capture the local and global normal patterns of those non-Euclidean data.

3.2 Related Work

Memory Networks Due to their ability to store and retrieve important information, memory networks have been proposed and successfully applied to a wide range of domains [83–88]. For generative models, external memory is exploited to store the local detail information [86] and prevent the model collapsing problem [87]. Considering that memory can be used to record the prototypical patterns of normal data, a number of studies [83, 84] proposed to augment AEs with a memory module for image or video anomaly detection. Despite the success of these methods, their memory networks are not applicable to GLAD as graph data is non-Euclidean and contains diverse graph structures and attributes where abnormality may exist. Our

hierarchical node-to-graph memory modules are specifically designed to address this problem.

3.3 Methodology

3.3.1 The GLAD Problem

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote a graph, where $\mathcal{V}$ is the set of N nodes and $\mathcal{E}$ is the set of edges. The graph structure $\mathcal{E}$ is commonly represented by an adjacency matrix $\mathbf{A} \in [0,1]^{N \times N}$ where $\mathbf{A}_{ij} = 1$ if node i and j are connected with an edge and $\mathbf{A}_{ij} = 0$ otherwise. If $\mathcal{G}$ is an attributed graph, the features associated with nodes can be represented as $\mathbf{X} \in \mathbb{R}^{N \times d}$ where d is the feature dimension. Therefore, a graph can also be denoted as $\mathcal{G} = (\mathbf{A}, \mathbf{X})$. This work targets graph-level anomaly detection. Specifically, given a set of K normal training graphs $\{\mathcal{G}_i = (\mathbf{A}_i, \mathbf{X}_i)\}_{i=1}^{K}$, we aim to learn an anomaly scoring function that assigns a high anomaly score to a test graph $\mathcal{G}$ if it significantly deviates from the majority in a set of graphs.

3.3.2 Overview of the Proposed Hierarchical Memory Networks

We introduce HimNet to learn hierarchical node and graph memory blocks that respectively capture local and global normal patterns for GLAD. HimNet consists of four key components, namely graph encoder, graph decoder, node and graph memory modules, as shown in Figure 3.1. The node memory module is designed as a three-dimensional tensor that consists of multiple two-dimensional memory blocks, with each block capturing one type of normal pattern on all nodes. On the other hand, the graph memory is designed as a two-dimensional matrix, with each memory block capturing normal patterns on graph-level representations. These two memory modules are trained to capture hierarchical normal patterns of graph data, enabling the detection of locally- and globally-anomalous graphs.

Given an input graph, the encoder learns the node-level representation, and

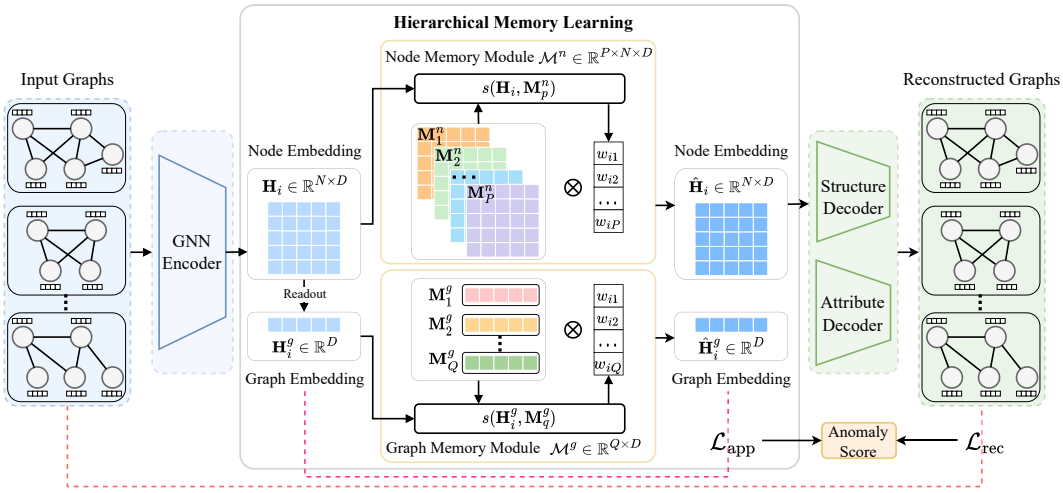


Figure 3.1 : Overview of the proposed method, HimNet. It learns a three-dimensional tensor-based node memory module $\mathcal{M}^n \in \mathbb{R}^{P\times N\times D}$ and a two-dimensional matrix-based graph memory module $\mathcal{M}^g \in \mathbb{R}^{Q\times D}$, where P and Q denotes the number of node and graph memory blocks respectively, D is the dimensionality of learned node representations, and N is the number of nodes.

graph-level representation is obtained by applying a readout function on it. Traditionally, the decoder takes the node-level representation as input to reconstruct the input graph. However, this would increase the probability of the graph autoencoder reconstructing the abnormal graphs well. To tackle this issue, HimNet decouples the decoder from the encoder by replacing the encoded node-level representation with a combination of local patterns stored in the node memory module. Moreover, the graph-level representation is approximated by global patterns in the graph memory module. Then, the proposed model is optimized by minimizing graph reconstruction error and graph approximation error. This not only optimizes the parameters of the encoder and decoder but also forces the two memory modules to learn prime patterns of normal training graphs at both node and graph levels. After model optimization, given a test graph sample, the decoder takes the local normal patterns in the node memory module as input and the graph-level representation is approximated by the

global normal patterns in the graph memory module. The graph reconstruction error together with the graph approximation error can then be used as an effective anomaly score.

3.3.3 Graph Autoencoder

In this chapter, we build HimNet using a graph autoencoder (GAE) [89] architecture to learn hierarchical memory modules. Before delving into the details of HimNet, we give an introduction to the graph autoencoder which consists of a GNN-based encoder and decoder.

Encoder

GNNs have recently emerged as a powerful class of deep-learning models for graph-structured data [20, 70, 90]. In this work, we employ GCN [20] as the graph encoder to generate the latent node-level and graph-level representations.

Let $\phi_e(\cdot : \Theta_e)$ be the GCN encoder parameterized by Θ_e. For every graph $\mathcal{G}_i = (\mathbf{A}_i, \mathbf{X}_i)$, the encoder takes the adjacency matrix $\mathbf{A}_i$ and node attributes $\mathbf{X}_i$ as input to learn the node embedding by transforming and aggregating its neighboring information. The formulation of the GCN encoder at the l-th layer can be expressed as follows:

$$\mathbf{H}_i^l = \mathrm{ReLU}(\hat{\mathbf{A}}_i \mathbf{H}_i^{(l-1)} \Theta_e^l)\,, \tag{3.1}$$

where $\mathbf{H}_i^l$ and Θ_e^l represent the node representations and weight parameters of the GCN encoder at the l-th layer, respectively, and $\mathrm{ReLU}(\cdot)$ is the non-linear activation function. $\hat{\mathbf{A}}_i = \tilde{\mathbf{D}}_i^{-\frac{1}{2}} \tilde{\mathbf{A}}_i \tilde{\mathbf{D}}_i^{-\frac{1}{2}}$, where $\tilde{\mathbf{A}}_i = \mathbf{A}_i + \mathbf{I}$ ($\mathbf{I}$ is an identity matrix) and $\tilde{\mathbf{D}}_i$ is the degree matrix of $\tilde{\mathbf{A}}_i$. $\mathbf{H}_i^{(l-1)}$ represents the node representation at the $(l-1)$-th layer and $\mathbf{H}_i^{(0)} = \mathbf{X}_i$. If the input graph $\mathcal{G}_i$ is a plain graph, the node degree is typically used as the attribute [26]. Assuming the output dimension of the encoder is D, the learned node representation can be formulated as $\mathbf{H}_i \in \mathbb{R}^{N \times D}$ where N is

the number of nodes in the graph.

To obtain graph-level representations, a readout function is commonly applied to the learned node representation $\mathbf{H}_i$. There are many readout functions, such as maxing, averaging, summation, and concatenation [2, 6]. In this chapter, we adopt the averaging function, which calculates the mean of node representations along the node dimension to get the graph-level representation $\mathbf{H}_i^g \in \mathbb{R}^D$. The resulting representation $\mathbf{H}_i^g$ captures the overall structural and semantic information of the graph $\mathcal{G}_i$.

Decoder

To accurately reconstruct the original graph $\mathcal{G}_i$, two decoders $\phi_d^s(\mathbf{H}_i)$ and $\phi_d^a(\mathbf{H}_i)$, which take node representation $\mathbf{H}_i$ as input, are employed to reconstruct the graph structure and node attribute respectively.

For the graph structure decoder $\phi_d^s(\mathbf{H}_i)$, we implement it as the inner product of the latent node representation $\mathbf{H}_i$ as follows:

$$\mathbf{A}_i^{'} = \sigma(\mathbf{H}_i\mathbf{H}_i^T)\,, \tag{3.2}$$

where $\mathbf{A}_i^{'}$ denotes the reconstructed graph structure, $\mathbf{H}_i^T$ is the transpose of $\mathbf{H}_i$, and $\sigma(\cdot)$ represents the activation function.

To reconstruct the node attribute, we use the GCN [20] as the attribute decoder $\phi_d^a(\mathbf{H}_i)$ and the formulation at the l-th layer can be expressed as:

$$\tilde{\mathbf{H}}_i^l = \mathrm{ReLU}(\hat{\mathbf{A}}_i\tilde{\mathbf{H}}_i^{(l-1)}\Theta_d^l)\,, \tag{3.3}$$

where $\tilde{\mathbf{H}}_i^l$ and Θ_d^l represent the latent node representations and weight parameters at l-th layer of the decoder respectively, with $\tilde{\mathbf{H}}_i^{(0)} = \mathbf{H}_i$. We denote the reconstructed node attribute as $\mathbf{X}_i^{'}$, which is obtained from the final layer of the decoder $\phi_d^a(\mathbf{H}_i)$.

For each input graph $\mathcal{G}_i = (\mathbf{A}_i, \mathbf{X}_i)$, GAE is optimized to minimize the reconstruction errors on the graph structures and node attributes:

$$\mathcal{L}_{\text{GAE}} = \|\mathbf{A}_i - \mathbf{A}_i^{'}\|_F^2 + \|\mathbf{X}_i - \mathbf{X}_i^{'}\|_F^2 \,, \tag{3.4}$$

where $\|\cdot\|_F$ represents a Frobenius norm.

By minimizing Eq. (3.4), GAE is driven to fit the patterns of normal training graph data and preserve the semantics of them. During inference, GAE would produce higher reconstruction errors for anomalous graphs than normal graphs, as the abnormal graphs are distinctive from the normal graphs and are not accessible to GAE during the training process. Therefore, the reconstruction error $\mathcal{L}_{GAE}$ can be directly used as the criterion for anomaly detection. However, solely relying on $\mathcal{L}_{GAE}$ often cannot yield satisfactory anomaly detection performance, as demonstrated in the experiments section. This is primarily because graph is difficult to reconstruct, leading to less discriminative power in differentiating normal and abnormal graphs.

Moreover, such a GAE cannot model graph-level patterns well, as graph representations are not explored in GAE. In this work, we propose to learn hierarchical memory modules to address this problem.

3.3.4 Hierarchical Memory Learning

Hierarchical memory learning consists of two memory modules: node and graph memory modules, which are designed to capture hierarchical node-to-graph patterns of the normal training graphs and facilitate the detection of graphs that are abnormal in part or in whole.

Graph Memory Module

The proposed graph module aims to capture the prototypical patterns inherent in the graph representations $\{\mathbf{H}_i^g\}_{i=1}^K$ (K is the number of training graphs) through

a set of graph memory blocks, denoted as $\mathcal{M}^g = \{\mathbf{M}_q^g \in \mathbb{R}^D\}_{q=1}^Q$, where Q is the total number of memory blocks and each block $\mathbf{M}_q^g$ is of the same dimensionality size as the graph representation $\mathbf{H}_i^g$.

Since the graph memory blocks capture prototypical patterns of graph representations, a graph representation $\mathbf{H}_i^g$ can be approximated using the following equation:

$$\hat{\mathbf{H}}_i^g = \sum_{q=1}^{Q} w_{iq}\mathbf{M}_q^g, \quad \text{s.t.} \quad \sum_{q=1}^{Q} w_{iq} = 1\,, \tag{3.5}$$

where $\hat{\mathbf{H}}_i^g$ is the approximated representation of $\mathbf{H}_i^g$ from the memory blocks, and w_{iq} is the weight of the memory block $\mathbf{M}_q^g$ for $\mathbf{H}_i^g$, with the summation of the weights constrained to be one. The weight w_{iq} reflects the correlation between each graph memory block and the graph representation, i.e., a higher correlation induces a larger weight. Therefore, to calculate w_{iq}, we first employ a cosine similarity function $s(\cdot)$ to measure the similarity between $\mathbf{M}_q^g$ and $\mathbf{H}_i^g$:

$$s(\mathbf{H}_i^g, \mathbf{M}_q^g) = \frac{\mathbf{H}_i^g (\mathbf{M}_q^g)^T}{\|\mathbf{H}_i^g\| \|\mathbf{M}_q^g\|}\,. \tag{3.6}$$

To impose the summation constraint, we further normalize the similarities via the following softmax operation to obtain the final weight:

$$w_{iq} = \frac{\exp(s(\mathbf{H}_i^g, \mathbf{M}_q^g))}{\sum_{q=1}^{Q} \exp(s(\mathbf{H}_i^g, \mathbf{M}_q^g))}\,. \tag{3.7}$$

After obtaining the approximated graph representation $\hat{\mathbf{H}}_i^g$, we calculate the approximation error via the following $\mathcal{L}_{\text{app}}$ loss:

$$\mathcal{L}_{\text{app}} = \|\mathbf{H}_i^g - \hat{\mathbf{H}}_i^g\|_F^2\,. \tag{3.8}$$

In the training phase, the optimization of Eq. (3.8) not only minimizes the approximation error through an efficient combination of the graph memory blocks but also forces the graph memory blocks to learn the most crucial patterns of the

graph representations. In this way, during the test phase, the approximation errors for normal and abnormal graphs would become distinct. This occurs because the approximated graph representation is constructed solely through the weighted combination of the learned normal patterns of graph representations.

Node Memory Module

Different from the graph memory module that captures the normal patterns at the graph-level representations, the node memory module is designed to capture the fine-grained, normal patterns on the node representations $\{\mathbf{H}_i \in \mathbb{R}^{N\times D}\}_{i=1}^{K}$. Specifically, the node memory module is designed as a three-dimensional tensor, consisting of P two-dimensional memory matrices, $\mathcal{M}^n = \{\mathbf{M}_p^n \in \mathbb{R}^{N\times D}\}_{p=1}^{P}$, with each memory block $\mathbf{M}_p^n$ having the same dimensionality size as the representations of all nodes. This way helps effectively capture interactions across all nodes and their local neighborhood.

To reduce the probability of the decoder reconstructing the abnormal graph unexpectedly, for a node representation $\mathbf{H}_i$, the node memory module approximates it with $\hat{\mathbf{H}}_i$ and feeds $\hat{\mathbf{H}}_i$ to the decoder. Formally, $\hat{\mathbf{H}}_i$ is obtained by:

$$\hat{\mathbf{H}}_i = \sum_{p=1}^{P} w_{ip}\mathbf{M}_p^n, \quad \text{s.t.} \quad \sum_{p=1}^{P} w_{ip} = 1\,, \tag{3.9}$$

where w_{ip} is the weight of the memory block $\mathbf{M}_p^n$ for $\mathbf{H}_i$ and the summation of w_{ip} is constrained to 1. To compute the value of w_{ip}, we adopt the same approach used in the graph memory block. Specifically, we first calculate the similarity between the node representation $\mathbf{H}_i$ and each node memory block $\mathbf{M}_p^n$. Then, we normalize the similarities through the softmax function to obtain the final weight value w_{ip}.

The approximated node representation $\hat{\mathbf{H}}_i$ is fed as the input to the graph structure decoder $\phi_d^s(\cdot)$ and node attribute decoder $\phi_d^a(\cdot)$. In this way, the reconstruction

error based on $\hat{\mathbf{H}}_i$ can be reformulated as:

$$\mathcal{L}_{\text{rec}} = \|\mathbf{A}_i - \phi_d^s(\hat{\mathbf{H}}_i)\|_F^2 + \|\mathbf{X}_i - \phi_d^a(\hat{\mathbf{H}}_i)\|_F^2 . \tag{3.10}$$

Compared to GAE which reconstructs the original graph depending on the encoded node representation, the node memory module performs graph construction solely based on the weighted combination of the node memory blocks. During training, the graph memory blocks are driven to learn the most representative patterns in the encoded node representations by minimizing the reconstruction error in Eq. (3.10). While in the testing phase, regardless of whether the input graph is normal or not, the decoder only takes different combinations of the learned normal patterns as input and outputs the normal-like graphs. Consequently, the reconstruction errors between normal and abnormal graphs would become significantly different. Overall, the node memory module decouples the decoder from the encoder, resulting in the graph reconstruction being more sensitive to the anomaly.

3.3.5 Training and Inference

Training Objective

By jointly employing the graph and node memory modules, HimNet aims to capture the hierarchical normal patterns of graphs. To achieve this goal, for each graph, our model is optimized by minimizing the combined objective of Eq. (3.8) and Eq. (3.10):

$$\mathcal{L}_{\text{rec}}^{'} = \|\mathbf{A}_i - \phi_d^s(\hat{\mathbf{H}}_i)\|_F^2 + \|\mathbf{X}_i - \phi_d^a(\hat{\mathbf{H}}_i)\|_F^2 + \|\mathbf{H}_i^g - \hat{\mathbf{H}}_i^g\|_F^2 . \tag{3.11}$$

To further enhance the discrimination of HimNet for normal and abnormal graphs, we adopt the hard shrinkage strategy [83] on the weight parameters w_{ip} and w_{iq}. Besides, the entropy of w_{ip} and w_{iq} is calculated and minimized during the

training, which can be formulated as follows:

$$\mathcal{L}_{\text{entropy}} = \sum_{i=1}^{P} -w_{ip} \log w_{ip} + \sum_{i=1}^{Q} -w_{iq} \log w_{iq} \,. \tag{3.12}$$

By employing the hard shrinkage and the entropy term, the weight parameters would become more sparse, i.e., the encoded node and graph representations are approximated with fewer memory blocks. This requires the chosen memory blocks to be more relevant to the encoded representations and also forces the memory blocks to learn more informative patterns.

The final training objective is obtained by combining Eq. (3.11) and Eq. (3.12):

$$\mathcal{L}_{\text{train}} = \mathcal{L}^{'}_{\text{rec}} + \alpha \mathcal{L}_{\text{entropy}} \,, \tag{3.13}$$

where α is a hyperparameter controlling the importance of the entropy term.

Inference

By optimizing Eq. (3.13), HimNet can capture the hierarchical normal patterns of graphs. As a result, for a normal graph, HimNet is capable of reconstructing it effectively with the memory blocks learned in both node and graph memory modules. However, for an abnormal graph, the value in Eq. (3.11) tends to be high. Therefore, we adopt the loss term Eq. (3.11) as the anomaly score, where a higher value indicates a larger probability of being an abnormal graph.

3.4 Experiments

3.4.1 Experimental Setups

Datasets

To verify the effectiveness of HimNet, we conduct experiments on 16 publicly available graph datasets from two popular application domains: i) biochemical molecules (PROTEINS_full, ENZYMES, AIDS, DHFR, BZR, COX2, DD, NCI1,

HSE, MMP, p53, PPAR-gamma, and hERG) and ii) social networks (IMDB, REDDIT, and COLLAB). The statistics of these graph datasets* are summarized in Table 3.1. Specifically, the first six datasets in Table 3.1 are attributed graphs and the other datasets consist of plain graphs. Moreover, HSE, MMP, p53, and PPAR-gamma contain real anomalies, while the other graph datasets are originally constructed for graph classification. Following [41, 46, 91, 92], these datasets are converted for GLAD by treating the minority class in these datasets as anomalies.

Table 3.1 : Key Statistics of Graph Datasets.

Dataset	**Category**	**# Graphs**	**# Avg.Nodes**	**# Avg.Edges**	**# Anomaly Rate**
PROTEINS_full	Biochemical Molecules	1,113	39.06	72.82	0.60
ENZYMES	Biochemical Molecules	600	32.63	62.14	0.17
AIDS	Biochemical Molecules	2,000	15.69	16.2	0.20
DHFR	Biochemical Molecules	467	42.43	44.54	0.61
BZR	Biochemical Molecules	405	35.75	38.36	0.79
COX2	Biochemical Molecules	467	41.22	43.45	0.78
DD	Biochemical Molecules	1,178	284.32	715.66	0.58
NCI1	Biochemical Molecules	4,110	29.87	32.3	0.50
HSE	Biochemical Molecules	8,417	16.89	17.23	0.04
MMP	Biochemical Molecules	7,558	17.62	17.98	0.16
p53	Biochemical Molecules	8,903	17.92	18.34	0.10
PPAR-gamma	Biochemical Molecules	8,451	17.38	17.72	0.06
hERG	Biochemical Molecules	655	26.48	28.79	0.31
IMDB	Social Networks	1,000	19.77	96.53	0.50
REDDIT	Social Networks	2,000	429.63	497.75	0.50
COLLAB	Social Networks	5,000	74.49	2,457.78	0.52

*All the graph datasets are available on `https://chrsmrrs.github.io/datasets/docs/datasets/` except hERG which is obtained from `https://tdcommons.ai/single_pred_tasks/tox/`

Baseline Methods

Several baseline methods from two categories are used for comparison to the proposed method. The first category consists of two-step methods that use state-of-the-art graph representation learning methods to extract graph representations and then apply an advanced anomaly detector on the learned representations to identify anomalous graphs. Specifically, we employ InfoGraph [39], Weisfeiler-Lehamn (WL) [17], and propagation kernel (PK) [37] as the graph encoder respectively and utilize isolation forest as the anomaly detector following [46]. The second category of baselines includes OCGCN [44], GLocalKD [46], and GAE [89] that are trained in an end-to-end manner. OCGCN [44] applied an SVDD objective on top of GCN-based representations for graph anomaly detection. GLocalKD [46] utilized random knowledge distillation to identify anomalies. GAE [89] used the graph reconstruction error to detect anomalous graphs.

Implementation Details

To ensure fair comparisons, we utilize a three-layer GCN [20] as the graph encoder following [46]. The dimensions of the latent layer and output feature are set to 512 and 256, respectively. The node attribute decoder is a two-layer GCN with the dimension of the latent layer set as 256. The batch size is 300 for all datasets except for HSE, MMP, p53, and PPAR-gamma, whose batch size is 2000 since these datasets contain more graphs. The hyperparameter α is set to 0.01 for all datasets. This work targets detecting anomalous graphs within multiple graphs. However, the number of nodes varies across graphs, which hinders the parallel processing of graph data. To address this issue, we augment the adjacency and the attribute matrices with zero padding to match the same size as the largest graph.

Evaluation

We employ the commonly used metric, area under receiver operating characteristic curve (AUC), to evaluate the anomaly detection performance. A higher AUC value indicates better performance. The mean and standard deviation of AUC results are reported by performing 5-fold cross-validation for all datasets except for HSE, MMP, p53 and PPAR-gamma which have widely-used predefined train and test splits. For these datasets, we report the results by running the experiments five times with different random seeds.

3.4.2 Comparison to State-of-the-art Methods

The AUC results of the proposed method and the baseline methods are reported in Table 3.2. From the average rank results in the table, we can see that end-to-end methods generally perform better than two-step methods, which highlights the significance of learning tailored representations for graph-level anomaly detection. Further, our method outperforms all the methods on 13 out of 16 datasets and achieves highly competitive performance in the remaining three datasets. In comparison to GAE [89], our method incorporates two memory modules to learn hierarchical normal patterns. The performance improvements over GAE [89] and other counterparts demonstrate the effectiveness of exploiting memory modules to capture hierarchical normal patterns for anomalous graph detection. Note that GAE performs poorly on NCI1 and REDDIT, while our method achieves very promising results. This demonstrates that the rich semantics learned in HimNet allow significantly better performance than GAE in distinguishing abnormal and normal graphs, especially when the graphs are large and difficult to reconstruct, e.g., those in REDDIT.

We also perform a paired Wilcoxon signed rank test [93] to verify the statistical significance of HimNet against the baselines across all 16 datasets and the results are shown in the bottom line of Table 3.2. We can see that our method surpasses

all baseline approaches with a confidence level greater than 98%.

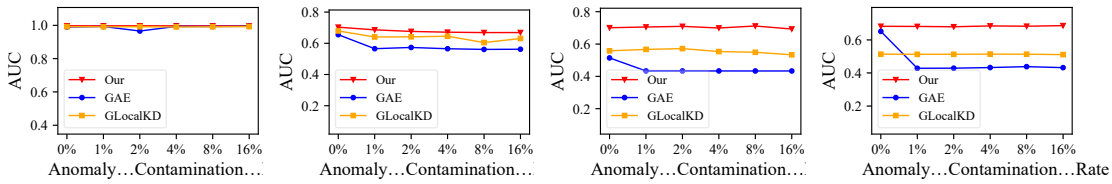
Table 3.2 : AUC results (in percent, mean±std) on 16 real-world graph datasets. The best and second performances in each row are boldfaced and underlined, respectively.

Dataset	**InfoGraph-iF**	**WL-iF**	**PK-iF**	**OCGCN**	**GLocalKD**	**GAE**	**Our**
PROTEINS_full	46.4±1.9	63.9±1.8	62.7±0.9	71.8±3.6	**78.5**±3.4	76.6±2.2	77.2±1.5
ENZYMES	48.3±2.7	49.8±2.9	49.3±1.3	61.3±8.7	**63.6**±6.1	51.6±2.7	58.9±7.6
AIDS	70.3±3.6	63.2±5.0	47.6±1.4	66.4±8.0	99.2±0.4	99.0±0.5	**99.7±0.3**
DHFR	48.9±1.5	46.6±1.3	46.7±1.3	49.5±8.0	55.8±3.0	51.4±3.6	**70.1±1.7**
BZR	52.8±6.0	53.3±3.2	52.5±5.2	65.8±7.1	67.9±6.5	65.5±8.3	**70.3±5.4**
COX2	58.0±5.2	53.2±2.7	51.5±3.6	62.8±7.2	58.9±4.5	58.7±4.9	**63.7±7.6**
DD	47.5±1.2	69.9±0.6	70.6±1.0	60.5±8.6	80.5±1.7	80.4±1.7	**80.6±2.1**
NCI1	49.4±0.9	54.5±0.8	53.2±0.6	62.7±1.5	68.3±1.5	35.0±1.9	**68.6±1.9**
HSE	48.4±2.6	47.7±0.0	48.9±0.3	38.8±4.1	59.1±0.1	59.0±0.5	**61.3±3.9**
MMP	53.9±2.2	47.5±0.0	48.8±0.2	45.7±3.8	67.6±0.1	67.3±0.4	**70.3±2.9**
p53	51.1±1.4	47.3±0.0	48.6±0.4	48.3±1.7	63.9±0.2	64.0±0.1	**64.6±0.2**
PPAR-gamma	52.1±2.3	51.0±0.0	49.9±1.7	43.1±4.3	64.4±0.1	65.8±1.6	**71.1±3.4**
hERG	60.7±3.3	66.5±4.2	67.9±3.4	56.9±4.9	70.4±4.9	68.1±8.7	**75.4±3.2**
IMDB	52.0±2.8	44.2±3.2	44.2±3.5	53.6±14.8	51.4±3.9	65.2±4.4	**68.3±3.2**
REDDIT	45.7±0.3	45.0±1.3	45.0±1.2	75.9±5.6	**78.2**±1.6	21.8±1.9	78.0±2.5
COLLAB	45.3±0.3	50.6±2.0	52.9±2.3	40.1±18.3	52.5±1.4	52.8±1.4	**55.3±3.2**
Avg.Rank	5.25	5.63	5.31	4.75	2.31	3.50	1.25
p-value	0.0004	0.0004	0.0004	0.0008	0.0113	0.0004	-

3.4.3 Robustness w.r.t Anomaly Contamination

This subsection evaluates the robustness of HimNet under different levels of anomaly contamination in training data. This scenario is generally very realistic since the graph data collected in the world may be contaminated by anomalies and noise. To simulate this setting, given the original training data that contains normal and abnormal data, instead of discarding abnormal data, we combine τ% of

the abnormal data with the normal training data to form the contaminated training data. Specifically, we vary the anomaly contamination rate τ from 0% to 16% and compare the performance of HimNet with the two best baseline methods – GLocalKD [46] and GAE [89] – as the baselines. Without loss of generality and due to the page limits, we perform experiments on four datasets, including three from biochemical molecules (AIDS, BZR, and DHFR) and one from social networks (IMDB).



(a) AIDS (b) BZR (c) DHFR (d) IMDB

Figure 3.2 : Results of GAE, GLocalKD, and HimNet under various contamination rates.

Figure 3.2 shows the AUC results of GAE, GLocalKD, and HimNet w.r.t. different anomaly contamination rates. Compared to the two baselines, our method achieves the best anomaly detection performance in all settings, particularly on DHFR and IMDB. We can also see that both GLocalKD and our method demonstrate consistent performance on all datasets for different anomaly contamination rates, while GAE experiences a significant performance drop with a slight increase in the number of anomalous training instances, except for the AIDS dataset. The reason for the superior and stable performance of our method is its ability to learn and store hierarchical patterns of the majority of the training data. As a result, anomalous graphs can be readily detected since they cannot be reconstructed effectively using the learned hierarchical memory blocks. Note that all three methods perform similarly on the AIDS dataset, which could be due to the distinguishability

between normality and abnormality being more apparent compared to the other datasets.

3.4.4 Ablation Study

We use the GAE as our base model to evaluate the importance of our proposed node and graph memory modules, which are the key driving components in HimNet. To verify the importance of each component, we conduct experiments on two variants of the proposed method, i.e., $\text{HimNet}_{\text{w/o node}}$ and $\text{HimNet}_{\text{w/o graph}}$ that respectively discard the node and graph memory module.

The results of HimNet, its two variants, and GAE are reported in Table 3.3. From the table, we can derive the following observations. First, by incorporating the node or graph memory module into GAE, the anomaly detection performance is significantly enhanced on nearly all datasets, which verifies the effectiveness of each of our proposed memory modules. Using GAE without memory modules is ineffective on some challenging datasets with large graphs and/or complex node attributes, such as DHFR and REDDIT. Second, $\text{HimNet}_{\text{w/o node}}$ and $\text{HimNet}_{\text{w/o graph}}$ perform differently across graph datasets, which indicates that the dominance of locally or globally anomalous graphs varies across the graph datasets. For example, $\text{HimNet}_{\text{w/o graph}}$ outperforms $\text{HimNet}_{\text{w/o node}}$ on NCI1, indicating the anomalous graphs are more dominated by locally anomalous graphs in NCI1. Third, the performance improvement over GAE is further boosted by the utilization of both node and graph memory modules. This demonstrates the importance of capturing the hierarchical normal patterns that enable the simultaneous detection of locally and globally anomalous graphs.

Table 3.3 : AUC performance of the proposed method and its variants.

Dataset	**GAE**	**HimNet w/o node**	**HimNet w/o graph**	**HimNet**
PROTEINS_full	76.6±2.2	76.4±2.5	76.3±3.1	**77.2±1.5**
ENZYMES	51.6±2.7	52.0±4.3	55.7±5.4	**58.9±7.6**
AIDS	99.0±0.5	99.3±0.4	99.6±0.3	**99.7±0.3**
DHFR	51.4±3.6	51.9±3.5	68.3±6.7	**70.1±1.7**
BZR	65.5±8.3	69.9±4.9	67.4±4.8	**70.3±5.4**
COX2	58.7±4.9	59.7±7.0	**64.7±5.9**	63.7±7.6
DD	80.4±1.7	80.2±1.8	80.4±2.0	**80.6±2.1**
NCI1	35.0±1.9	33.9±5.7	62.2±2.4	**68.6±1.9**
HSE	59.0±0.5	59.4±0.4	61.6±4.3	**61.3±3.9**
MMP	67.3±0.4	68.7±0.8	69.1±0.2	**70.3±2.9**
p53	64.0±0.1	64.4±0.1	**64.9±0.8**	64.6±0.1
PPAR-gamma	65.8±1.6	67.6±0.1	67.5±3.9	**71.1±3.4**
hERG	68.1±8.7	68.7±0.8	73.2±3.2	**75.4±3.2**
IMDB	65.2±4.4	65.2±4.4	66.0±3.0	**68.3±3.2**
REDDIT	21.8±1.9	21.9±2.4	31.1±8.9	**78.0±2.5**
COLLAB	52.8±1.4	52.2±1.6	51.7±1.6	**55.3±3.2**

3.4.5 Analysis of Hyperparameters

We examine the sensitivity of our method, HimNet, w.r.t the number of memory blocks in the node and graph memory modules. Specifically, for one memory module, we fix the number of memory blocks to one and vary the number of memory blocks in the other memory module across $\{1, 2, 3, 4, 5, 6\}$. The results of all graph datasets are reported in Fig. 3.3. The results show that even using one memory block in each memory module, the proposed method can still achieve promising performance on some datasets, such as DD, DHFR, REDDIT, and hERG. This may be because the normal graphs in these datasets are more homogeneous and deviate from the abnormal graphs distinctly. HimNet is generally more robust to the number of graph memory blocks than the number of node memory blocks, except on the AIDS, BZR,

NCI1, and REDDIT datasets. Also, increasing the number of memory blocks does not always bring better results. In some cases, it can even degrade the detection performance. This is mainly because the larger memory modules may boost the expressiveness of memory modules, leading to the failure cases where the abnormal graphs can also be well reconstructed.

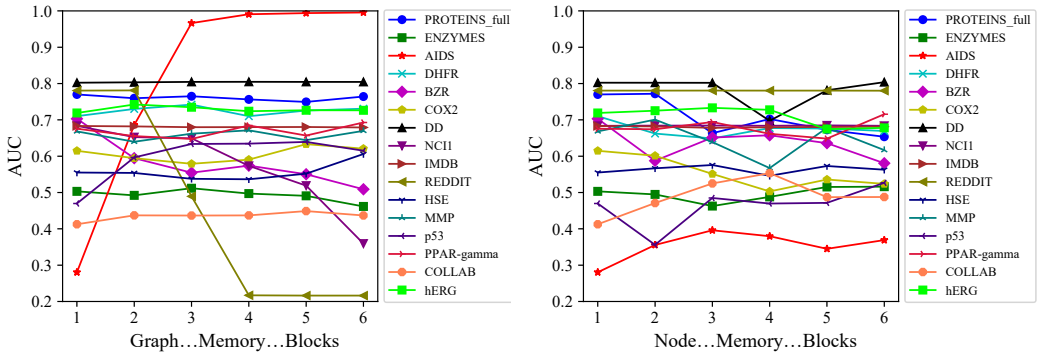


Figure 3.3 : Results of HimNet w.r.t different numbers of graph and node memory blocks.

3.5 Conclusion

This chapter proposes hierarchical memory networks (HimNet) via a graph autoencoder architecture to learn hierarchical node and graph memory modules for graph-level anomaly detection. The node and graph memory modules explicitly capture the hierarchical normal patterns of graphs by jointly minimizing graph reconstruction and graph approximation errors. The learned hierarchical memory blocks enable effective detection of both locally- and globally-anomalous graphs. Extensive experiments on 16 datasets from different domains demonstrate the superiority of HimNet in detecting anomalous graphs compared to state-of-the-art methods. Furthermore, HimNet achieves promising performance even when the training data is largely contaminated by anomaly graphs, which shows its applicability in real-world applications with unclean training data.

Chapter 4

Zero-shot Generalist Graph Anomaly Detection with Unified Neighborhood Prompts

4.1 Introduction

Graph anomaly detection (GAD) aims to identify anomalous nodes that exhibit significant deviations from the majority of nodes in a graph. GAD has attracted extensive research attention in recent years [8,9,94] due to the broad applications in various domains such as spam review detection in online shopping networks [95,96] and malicious user detection in social networks [97]. To handle high-dimensional node attributes and complex structural relations between nodes, graph neural networks (GNNs) [6,98] have been widely exploited for GAD due to their strong ability to integrate the node attributes and graph structures. These methods can be roughly divided into two categories, *i.e.*, supervised and unsupervised methods. The former formulates GAD as a binary classification problem and aims to capture anomaly patterns under the guidance of labels [35,36,99]. By contrast, due to the difficulty of obtaining these class labels, the latter category takes the unsupervised approach that aims to learn normal graph patterns, *e.g.*, via data reconstruction or other proxy learning tasks that are related to GAD [28,30,32,100].

Despite their remarkable detection performance, these methods need to train a dataset-specific GAD model for each graph dataset. This one-model-for-one-dataset paradigm limits their applicability in real-world scenarios since training a model from scratch can incur extensive computation costs and require a large amount of labeled data for supervised GAD methods on each dataset [9, 52]. Training on a target

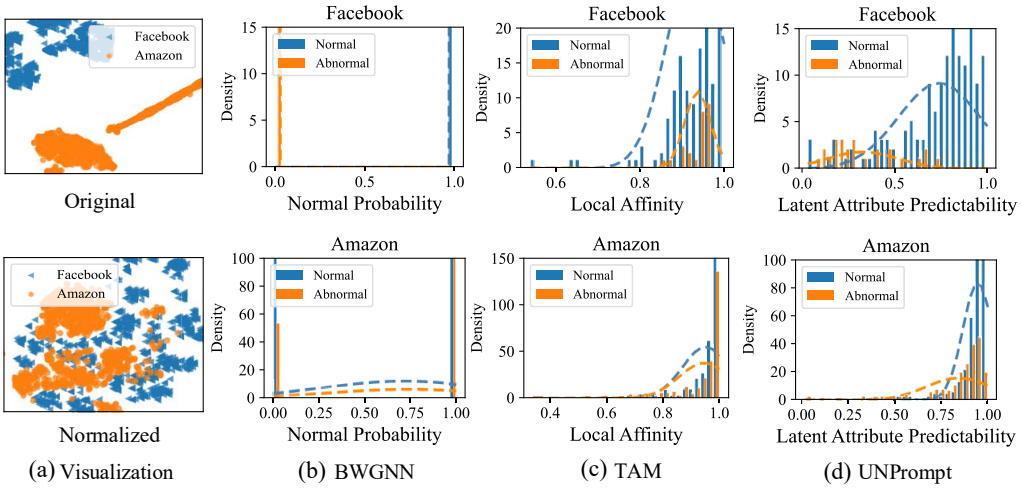


Figure 4.1 : **(a)** Visualization of two popular GAD datasets: Facebook and Amazon, where the node attributes are unified into a common semantic space via our proposed normalization compared to the original heterogeneous raw attributes. **(b)-(d)** The anomaly scores of BWGNN (normal probability), TAM (local affinity) and UNPrompt (latent attribute predictability) on the two datasets, where the methods are all trained on Facebook and tested on Amazon under the zero-shot setting. It is clear that BWGNN and TAM struggle to generalize from Facebook to Amazon, while UNPrompt can learn well to generalize across the datasets.

graph may even not be possible due to data privacy protection and regulation. To address this limitation, a new one-for-all anomaly detection (AD) paradigm, called generalist anomaly detection [50, 51], has been proposed for image AD with the emergence of foundation models such as CLIP [101]. This new direction aims to learn a generalist detection model on auxiliary datasets so that it can generalize to detect anomalies effectively in diverse target datasets without any re-training or fine-tuning. This chapter explores this direction in the area of GAD.

Compared to image AD, there are some unique challenges for learning generalist models for GAD. First, unlike image data where raw features are in the same RGB space, the node attributes in graphs from different applications and domains can differ significantly in node attribute dimensionality and semantics. For example, as a shopping network dataset, Amazon contains the relationships between users and reviews, and the node attribute dimensionality is 25. Differently, Facebook, a social network dataset, describes relationships between users with 576-dimensional attributes. Second, generalist AD models on image data rely on the superior generalizability learned in large visual-language models (VLMs) through pre-training on web-scale image-text-aligned data [50, 51], whereas there are no such foundation models for graphs [102]. Therefore, the key question here is: *can we learn generalist models for GAD on graph data with heterogeneous node attributes and structure without the support of foundation models?*

To address these challenges, we propose **UNPrompt**, a novel generalist GAD approach that learns *Unified Neighborhood Prompts* on a single auxiliary graph dataset and then effectively generalizes to directly detect anomalies in other graph datasets under a **zero-shot** setting. The key insight in UNPrompt is that i) the predictability of latent node attributes can serve as a generalized anomaly measure and ii) graph-agnostic normal and abnormal patterns can be learned via latent node attribute prediction in a properly normalized node attribute space. UNPrompt achieves this

through two main modules, including *coordinate-wise normalization-based node attribute unification* and *neighborhood prompt learning*. The former module aligns the dimensionality of node attributes across graphs and transforms the semantics into a common space via coordinate-wise normalization, as shown in Figure 4.1(a). In this way, the diverse distributions of node attributes are calibrated into the same semantic space. On the other hand, the latter module focuses on modeling graph-agnostic normal and abnormal patterns across different graph datasets. This is achieved in UNPrompt by learning generalized graph prompts in the normalized attributes of the neighbors of a target node via a latent node attribute prediction task. In doing so, UNPrompt, being trained on a small GNN using a single graph, can yield effective anomaly scores for detecting anomalous nodes in diverse unseen graphs without any re-training at the inference stage, as shown in Figure 4.1(b)-(d).

4.2 Related Work

Graph Prompt Learning. Prompt learning, initially developed in natural language processing, seeks to adapt large-scale pre-trained models to different downstream tasks by incorporating learnable prompts while keeping the pre-trained models frozen [103]. Specifically, it designs task-specific prompts capturing the knowledge of the corresponding tasks and enhances the compatibility between inputs and pre-trained models to improve the pre-trained models in downstream tasks. Recently, prompt learning has been explored in graphs to unify multiple graph tasks [104,105] or improve the transferability of graph models on the datasets across the different domains [106,107] which optimize the prompts with labeled data of various downstream tasks [105,108]. Although being effective in popular graph learning tasks like node classification and link prediction, they are inapplicable to generalist GAD due to the unsupervised nature and/or irregular distributions of anomalies.

4.3 Methodology

4.3.1 Preliminaries

Notations. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an attributed graph with N nodes, where $\mathcal{V} = \{v_1, v_2, \ldots, v_N\}$ represents the node set and $\mathcal{E}$ is the edge set. The attributes of nodes can be denoted as $X = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]^T \in \mathbb{R}^{N \times d}$ and the edges between nodes can be presented by an adjacency matrix $A \in \{0, 1\}^{N \times N}$ with $A_{ij} = 1$ if there is an edge between v_i and v_j and $A_{ij} = 0$ otherwise. For simplicity, the graph can be represented as $\mathcal{G} = (A, X)$. In GAD, the node set can be divided into a set of normal nodes $\mathcal{V}_n$ and a set of anomalous nodes $\mathcal{V}_a$. Typically, the number of normal nodes is significantly larger than the anomalous nodes, *i.e.*, $|\mathcal{V}_n| \gg |\mathcal{V}_a|$. Moreover, the anomaly labels can be denoted as $\mathbf{y} \in \{0, 1\}^N$ with $\mathbf{y}_i = 1$ if $v_i \in \mathcal{V}_a$ and $\mathbf{y}_i = 0$ otherwise.

Conventional GAD. Conventional GAD typically focuses on model training and anomaly detection on the same graph. Specifically, given a graph $\mathcal{G}$, an anomaly scoring model $f : \mathcal{G} \to \mathbb{R}$ is optimized on $\mathcal{G}$ in a supervised or unsupervised manner. Then, the model is used to detect anomalies within the same graph. The model is expected to generate higher anomaly scores for abnormal nodes than normal nodes, *i.e.*, $f(v_i) < f(v_j)$ if $v_i \in \mathcal{V}_n$ and $v_j \in \mathcal{V}_a$.

Generalist GAD. Generalist GAD aims to learn a generalist model f on a single training graph so that f can be directly applied to different target graphs across diverse domains without any fine-tuning or re-training. More specifically, the model is optimized on $\mathcal{G}_{\text{train}}$ with the corresponding anomaly labels $\mathbf{y}_{\text{train}}$. After model optimization, the learned f is utilized to detect anomalies within different unseen target graphs $\mathcal{T}_{\text{test}} = \{\mathcal{G}_{\text{test}}^{(1)}, \ldots, \mathcal{G}_{\text{test}}^{(n)}\}$ which has heterogeneous attributes and/or structure to $\mathcal{G}_{\text{train}}$, *i.e.*, $\mathcal{G}_{\text{train}} \cap \mathcal{T}_{\text{test}} = \emptyset$. Depending on whether labeled nodes of

the target graph are provided during inference, the generalist GAD problem can be further divided into two categories, *i.e.*, **few-shot** and **zero-shot** settings. We focus on the zero-shot setting where the generalist models cannot get access to any labeled data of the testing graphs during both training and inference.

4.3.2 Overview of UNPrompt

The framework of UNPrompt is illustrated in Figure 4.2, which consists of two main modules, coordinate-wise normalization-based node attribute unification and neighborhood prompt learning. For all graphs, the attribute unification aligns the dimensionality of node attributes and transforms the semantics into a common space. Then, in the normalized space, the generalized latent attribute prediction task is performed with the neighborhood prompts to learn generalized GAD patterns. Specifically, UNPrompt aims to maximize the predictability of the latent attributes of normal nodes while minimizing those of abnormal nodes. In this chapter, we evaluate the predictability via the similarity. In doing so, the graph-agnostic normal and abnormal patterns are incorporated into the prompts. During inference, the target graph is directly fed into the learned models after node attribute unification without any re-training or labeled nodes of the graph. For each node, the predictability of latent node attributes is directly used as the normal score for final anomaly detection.

4.3.3 Node Attribute Unification

Graphs from different distributions and domains significantly differ in the dimensionality and semantics of node attributes. The premise of developing a generalist GAD model is to unify the dimensionality and semantics of node attributes into the same space. To address this issue, we propose a simple yet effective node attribute unification module which consists of feature projection and coordinate-wise normalization. Different from ARC [52] which aligns the attributes based on feature

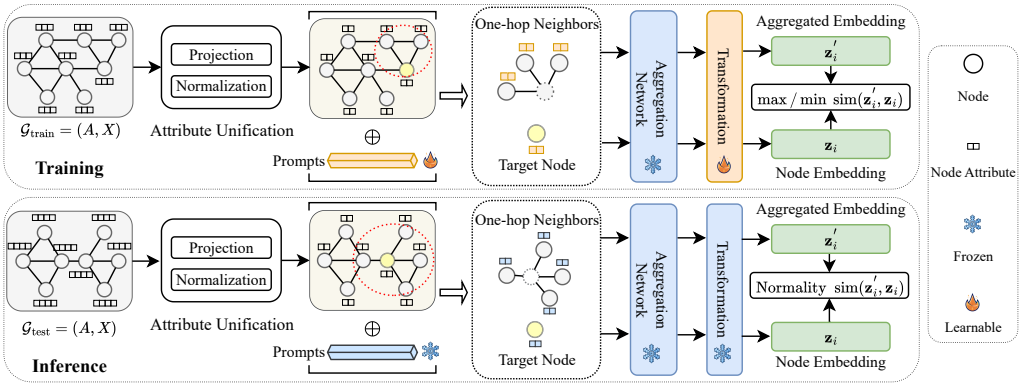


Figure 4.2 : Overview of UNPrompt. During training, the neighborhood prompts are optimized to capture generalized patterns by maximizing the predictability of the latent attributes of normal nodes while minimizing that of abnormal nodes. At the inference stage, the learned prompts are directly applied to the testing nodes, and the latent attribute predictability is used for GAD.

reordering using feature smoothness, we calibrate the feature distributions of diverse graphs into the same frame, resulting in a simpler yet effective alignment.

Feature Projection. To address the inconsistent attribute dimensions across graphs, various feature projection methods can be utilized, such as singular value decomposition [109] (SVD) and principal component analysis [110] (PCA). Formally, given the attribute matrix $X^{(i)} \in \mathbb{R}^{N^{(i)} \times d^{(i)}}$ of any graph $\mathcal{G}^{(i)}$ from $\mathcal{G}_{\text{train}} \cup \mathcal{T}_{\text{test}}$, we transform it into $\tilde{X}^{(i)} \in \mathbb{R}^{N^{(i)} \times d'}$ with the common dimensionality of d',

$$X^{(i)} \in \mathbb{R}^{N^{(i)} \times d^{(i)}} \xrightarrow[\text{Projection}]{\text{Feature}} \tilde{X}^{(i)} \in \mathbb{R}^{N^{(i)} \times d'} . \tag{4.1}$$

Coordinate-wise Normalization. Besides the inconsistent attribute dimensionality, the divergent node attribute semantics and distributions of different graphs pose significant challenges for generalist GAD. A recent study [111] has demonstrated that semantic differences across datasets are mainly reflected in the distri-

bution shifts and calibrating the distributions into a common frame helps learn more generalized AD models. Inspired by this, we propose to use coordinate-wise normalization to align the semantics and unify the distributions across graphs. Specifically, the transformed attribute matrix $\tilde{X}^{(i)}$ is shifted and re-scaled to have mean zeros and variance ones via the following equation:

$$\bar{X}^{(i)} = \frac{\tilde{X}^{(i)} - \boldsymbol{\mu}^{(i)}}{\boldsymbol{\sigma}^{(i)}}, \tag{4.2}$$

where $\boldsymbol{\mu}^{(i)} = [\mu_1^{(i)}, \ldots, \mu_{d'}^{(i)}]$ and $\boldsymbol{\sigma}^{(i)} = [\sigma_1^{(i)}, \ldots, \sigma_{d'}^{(i)}]$ are the coordinate-wise mean and variance of $\tilde{X}^{(i)}$ of the graph $\mathcal{G}^{(i)}$. In this way, the distributions of normalized attributes along each dimension are the same within and across graphs, as shown in Figure 4.1(a). This helps to capture the generalized normal and abnormal patterns for generalist GAD.

4.3.4 Neighborhood Prompt Learning

Latent Node Attribute Predictability as Anomaly Score. In this chapter, we reveal that the predictability of latent node attributes can serve as a generalized anomaly measure, and highly generalized normal and abnormal graph patterns can be learned via latent node attribute prediction in the normalized attribute space with neighborhood prompts. The key intuition of this anomaly measure is that normal nodes tend to have more connections with normal nodes of similar attributes due to prevalent graph homophily relations, resulting in a more homogeneous neighborhood in the normal nodes [32]. By contrast, the presence of anomalous connections and/or attributes makes abnormal nodes deviate significantly from their neighbors. Therefore, for a target node, its latent attributes (*i.e.*, node embedding) is more predictable based on the latent attributes of its neighbors if the node is a normal node, compared to abnormal nodes. The neighborhood-based latent attribute prediction is thus used to measure the normality for GAD. As shown in our experiments (see Figures 4.1(b)-(d) and Tables 4.2), it is a generalized anomaly measure that works

effectively across graphs. However, due to the existence of irrelevant and noisy attribute information in the original attribute space, the attribute prediction is not as effective as expected in the simply projected space. To address this issue, we propose to learn discriminative prompts via the latent attribute prediction task to enhance the effectiveness of this anomaly measure.

To achieve this, we first design a simple GNN g, a neighborhood aggregation network, to generate the aggregated neighborhood embedding of each target node. Specifically, given a graph $\mathcal{G} = (A, \bar{X})$, the aggregated neighborhood embeddings for each node are obtained as follows:

$$\tilde{Z} = g(\mathcal{G}) = \tilde{A}\bar{X}W\,, \tag{4.3}$$

where $\tilde{Z}$ is the aggregated representation of neighbors, $\tilde{A} = (D)^{-1}A$ is the normalized adjacency matrix with D being a diagonal matrix and its elements $D_{kk} = \sum_j A_{kj}$, and W is the learnable parameters. Compared to conventional GNNs such as GCN [98] and SGC [112], we do not require $\tilde{A}$ to be self-looped and symmetrically normalized as we aim to obtain the aggregated representation of all the neighbors for each node. To design the latent node attribute prediction task, we further obtain the latent attributes of each node as follows:

$$Z = \bar{X}W\,, \tag{4.4}$$

where Z serves as the prediction ground truth for the latent attribute prediction task. The adjacency matrix A is discarded to avoid carrying neighborhood-based attribute information into Z which would lead to ground truth leakage in this prediction task. We further propose to utilize the cosine similarity to measure this neighborhood-based latent attribute predictability for each node:

$$s_i = \text{sim}(\mathbf{z}_i, \tilde{\mathbf{z}}_i) = \frac{\mathbf{z}_i(\tilde{\mathbf{z}}_i)^T}{\|\mathbf{z}_i\|\|\tilde{\mathbf{z}}_i\|}\,, \tag{4.5}$$

where $\mathbf{z}$ and $\tilde{\mathbf{z}}_i$ are the i-th node embeddings in Z and $\tilde{Z}$ respectively. A higher

similarity denotes the target node can be well predicted by its neighbors and indicates the target is normal with a higher probability. Therefore, we directly utilize the similarity to measure the normal score of the nodes.

Neighborhood Prompting via Latent Attribute Prediction. After the pre-training, we aim to further learn more generalized normal and abnormal patterns via prompt tuning. Thus, we devise learnable prompts appending to the attributes of the neighboring nodes of the target nodes, namely *neighborhood prompts*, for learning robust and discriminative patterns that can detect anomalous nodes in different unseen graphs without any re-training during inference.

Specifically, neighborhood prompting aims to learn some prompt tokens that help maximize the neighborhood-based latent prediction of normal nodes while minimizing that of abnormal nodes simultaneously. To this end, the prompt is designed as a set of shared and learnable tokens that can be incorporated into the normalized node attributes. Formally, the neighborhood prompts are represented as $P = [\mathbf{p}_1, \ldots, \mathbf{p}_k]^T \in \mathbb{R}^{K \times d'}$ where K is the number of vector-based tokens $\mathbf{p}_i$. For each node in $\mathcal{G} = (A, \bar{X})$, the node attributes in the unified feature space are augmented by the weighted combination of these tokens, with the weights obtained from K learnable linear projections:

$$\hat{\mathbf{x}}_i = \bar{\mathbf{x}}_i + \sum_{j}^{K} \alpha_j \mathbf{p}_j \,, \quad \alpha_j = \frac{e^{(\mathbf{w}_j)^T \mathbf{x}_i^t}}{\sum_l^K e^{(\mathbf{w}_l)^T \mathbf{x}_i^t}} \,, \tag{4.6}$$

where α_j denotes the importance score of the token $\mathbf{p}_j$ in the prompt and $\mathbf{w}_j$ is a learnable projection. For convenience, we denote the graph modified by the graph prompt as $\tilde{\mathcal{G}} = (A, \bar{X} + P)$. Then, $\tilde{\mathcal{G}}$ is fed into the frozen pre-trained model g to obtain the corresponding aggregated embeddings $\tilde{Z}$ and node latent attributes Z via Eq.(4.3) and Eq.(4.4) respectively to measure the attribute predictability. To further enhance the representation discrimination, a transformation layer h is applied on

the learned $\tilde{Z}$ and Z to transform them into a more anomaly-discriminative feature space,

$$\tilde{Z} = h(\tilde{Z})\,, \quad Z = h(Z)\,. \tag{4.7}$$

The transformed representations are then used to measure the latent node attribute predictability with Eq.(4.5). To optimize P and h, we employ the following training objective,

$$\min_{P,h} \sum \ell(\mathbf{z}_i, \tilde{\mathbf{z}}_i)\,, \tag{4.8}$$

where $\ell(\mathbf{z}_i, \tilde{\mathbf{z}}_i) = -\text{sim}(\mathbf{z}_i, \tilde{\mathbf{z}}_i)$ if $\mathbf{y}_i = 0$, and $\ell(\mathbf{z}_i, \tilde{\mathbf{z}}_i) = \text{sim}(\mathbf{z}_i, \tilde{\mathbf{z}}_i)$ if $\mathbf{y}_i = 1$.

4.3.5 Pre-training of Neighborhood Aggregation Networks

We pre-train the neighborhood aggregation network g via graph contrastive learning [113] for subsequent graph prompt learning so that the generic normality and abnormality can be captured in the prompts. Without pre-training, the dataset-specific knowledge would be captured by the model if it is directly optimized based on the neighborhood-based latent attribute prediction of normal and abnormal nodes, limiting the generalizability of the model to other graphs.

Specifically, given the training graph $\mathcal{G} = (A, X)$, to construct contrastive views for graph contrastive learning, two widely used graph augmentations are employed, *i.e.*, edge removal and attribute masking [113]. The edge removal randomly drops a certain portion of existing edges in $\mathcal{G}$ and the attribute masking randomly masks a fraction of dimensions with zeros in node attributes, *i.e.*,

$$\hat{A} = A \circ R\,, \quad \hat{X} = [\mathbf{x}_1 \circ \mathbf{m}, \ldots, \mathbf{x}_N \circ \mathbf{m}]^T\,, \tag{4.9}$$

where $R \in \{0,1\}^{N \times N}$ is the edge masking matrix whose entry is drawn from a Bernoulli distribution controlled by the edge removal probability, $\mathbf{m} \in \{0,1\}^d$ is the attribute masking vector whose entry is independently drawn from a Bernoulli distribution with the attribute masking ratio, and $\circ$ denotes the Hadamard product.

By applying the graph augmentations to the original graph, the corrupted graph $\hat{\mathcal{G}} = (\hat{A}, \hat{X})$ forms the contrastive view for the original graph $\mathcal{G} = (A, X)$. Then, $\hat{\mathcal{G}}$ and $\mathcal{G}$ are fed to the shared model g followed by the non-linear projection to obtain the corresponding node embeddings, *i.e.*, $\hat{Z}'$ and Z'. For graph contrastive learning, the embeddings of the same node in different views are pulled closer while the embeddings of other nodes are pushed apart. The pairwise objective for each node pair $(\hat{\mathbf{z}}'_i, \mathbf{z}'_i)$ can be formulated as:

$$\ell(\hat{\mathbf{z}}'_i, \mathbf{z}'_i) = -\log \frac{e^{\text{sim}(\hat{\mathbf{z}}'_i, \mathbf{z}'_i)/\tau}}{e^{\text{sim}(\hat{\mathbf{z}}'_i, \mathbf{z}'_i)/\tau} + \sum_{j \neq i}^{N} e^{\text{sim}(\hat{\mathbf{z}}'_i, \mathbf{z}'_j)/\tau} + \sum_{j \neq i}^{N} e^{\text{sim}(\hat{\mathbf{z}}'_i, \hat{\mathbf{z}}'_j)/\tau}}, \tag{4.10}$$

where $\text{sim}(\cdot)$ represents the cosine similarity and τ is a temperature hyperparameter. Therefore, the overall objective can be defined as follows:

$$\mathcal{L}_{\text{contrast}} = \frac{1}{2N} \sum_{i=1}^{N} (\ell(\hat{\mathbf{z}}'_i, \mathbf{z}'_i) + \ell(\mathbf{z}'_i, \hat{\mathbf{z}}'_i)). \tag{4.11}$$

With the objective Eq.(4.11), the model g is optimized to learn transferable discriminative representations of nodes.

4.3.6 Training and Inference of UNPrompt

Training. Given $\mathcal{G}_{\text{train}}$, a neighborhood aggregation network g is optimized via graph contrastive learning. Then, the neighborhood prompts P and the transformation layer h are optimized to capture the graph-agnostic normal and abnormal patterns while keeping the pre-trained model g frozen. In this way, the transferable knowledge of the pre-trained g is maintained, while the neighborhood prompt learning helps learn the generalized normal and abnormal patterns.

Inference. During inference, given $\mathcal{G}^{(i)}_{\text{test}} \in \mathcal{T}_{\text{test}}$, the node attributes are first aligned. Then, the test graph $\mathcal{G}^{(i)}_{\text{test}}$ is augmented with the learned neighborhood prompt P and fed into the model g and the transformation layer h to obtain the neighborhood aggregated representations and the latent node attributes. Finally, the similarity (Eq.(4.5)) is used as the normal score for anomaly detection. Note that the inference

does not require any further re-training and labeled nodes of $\mathcal{G}_{\text{test}}^{(i)}$. The training and inference processes of UNPrompt are summarized in Algorithms 1 and Algorithm 2, respectively.

Algorithm 1 Training of UNPrompt

Input: Training graph $\mathcal{G}_{\text{train}} = (A, X)$; training epoch E

Output: Neighborhood aggregation network g, graph prompts $P = [\mathbf{p}_1, \ldots, \mathbf{p}_K]$, and transformation h.

Perform feature unification of X.

Pre-train g on $\mathcal{G}_{\text{train}}$ with graph contrastive learning in Eq.(4.11).

Keep model g frozen.

for $e = 1, \ldots, E$ **do**

    Obtain modified node attribute with prompts via Eq.(6).

    Obtain the neighborhood aggregated representation $\tilde{Z}$ via Eq.(3).

    Obtain the node representations Z via Eq.(4).

    Transform $\tilde{Z}$ and Z with h via Eq.(7).

    Optimize P and h by minimizing Eq.(8).

end for

4.3.7 Time Complexity Analysis

In this section, we analyze the time complexity of training UNPrompt. As discussed above, UNPrompt first pre-trains the aggregation network with graph contrastive learning. Then, the model remains frozen when optimizing neighborhood graph prompts and the transformation layer to capture the generalized normal and abnormal graph patterns. In the experimental section, we employ a one-layer aggregation network, denoting the number of hidden units as d_h. The time complexity of the graph contrastive learning is $\mathcal{O}(4E_1(|A|d_h + Nd_hd^{'} + 6Nd_h^2))$, where $|A|$ returns of the number of edges of the $\mathcal{G}_{\text{train}}$, N is the number of nodes, $d^{'}$ represents the predefined dimensionality of node attributes, and E_1 is the number of

Algorithm 2 Inference of UNPrompt

Input: Testing graphs $\mathcal{T}_{\text{test}} = \{\mathcal{G}_{\text{test}}^{(1)}, \ldots, \mathcal{G}_{\text{test}}^{(n)}\}$, neighborhood aggregation network g, graph prompts $P = [\mathbf{p}_1, \ldots, \mathbf{p}_K]$, and transformation h.
Output: Normal score of testing nodes.
for $\mathcal{G}_{\text{test}}^{(i)} = (A^{(i)}, X^{(i)}) \in \mathcal{T}_{\text{test}}$ **do**
    Perform feature unification of $X^{(i)}$.
    Obtain modified node attribute with prompts via Eq.(6).
    Obtain the neighborhood aggregated representation $\tilde{Z}^{(i)}$ via Eq.(3).
    Obtain the node representations $Z^{(i)}$ via Eq.(4).
    Transform $\tilde{Z}^{(i)}$ and $Z^{(i)}$ with h via Eq.(7).
    Obtain the normal score of nodes via Eq.(5).
end for

training epoch. After that, we freeze the learned model and learn the learnable neighborhood prompt tokens and the transformation layer to capture the shared anomaly patterns. In our experiments, we set the size of each graph prompt to K and implement the classification head as a single-layer MLP with the same hidden units d_h. Given the number of the training epoch E_2, the time complexity of optimizing the graph prompt and the transformation layer is $\mathcal{O}((4KNd^{'} + 2Nd_h^2)E_2)$, which includes both the forward and backward propagation. Note that, despite the neighborhood aggregation model being frozen, the forward and backward propagations of the model are still needed to optimize the task-specific graph prompts and the transformation layer. Therefore, the overall time complexity of UNPrompt is $\mathcal{O}(4E_1(|A|d_h + Nd_hd^{'} + 6Nd_h^2) + 2E_2(|A|d_h + Nd_hd^{'} + 2KNd^{'} + Nd_h^2))$, which is linear to the number of nodes, the number of edges, and the number of node attributes of the training graph. Note that, after the training, the learned generalist model is directly utilized to perform anomaly detection on various target graphs without any further training.

4.3.8 Unsupervised GAD with UNPrompt

To demonstrate the wide applicability of the proposed method, UNPrompt, we further perform unsupervised GAD with UNPrompt, which focuses on detecting anomalous nodes within one graph and does not have access to any node labels during training.

Specifically, we adopt the same pipeline in the generalist GAD setting, *i.e.*, graph contrastive pertaining and neighborhood prompt learning. Since we focus on anomaly detection on each graph separately, the node attribute unification is discarded for unsupervised GAD. However, the absence of node labels poses a challenge to learning meaningful neighborhood prompts for anomaly detection. To overcome this issue, we propose to utilize the pseudo-labeling technique to guide the prompt learning. Specifically, the normal score of each node is calculated by the neighborhood-based latent attribute predictability after the graph contrastive learning process:

$$s_i = \text{sim}(\mathbf{z}_i, \tilde{\mathbf{z}}_i)\,, \tag{4.12}$$

where $\mathbf{z}_i$ is the node representation learned by graph contrastive learning and $\tilde{\mathbf{z}}_i$ is the corresponding aggregated neighborhood representation. Higher s_i of node v_i typically indicates a higher probability of v_i being a normal node. Therefore, more emphasis should be put on high-score nodes when learning neighborhood prompts. To achieve this, the normal score s_i is transformed into the loss weight $w_i = \text{Sigmoid}(\alpha(s_i - t))$ where t is a threshold and α is the scaling parameter. In this way, w_i would approach 1 if $s_i > t$ and 0 otherwise. Overall, the objective for unsupervised GAD using UNPrompt can be formulated as follows:

$$\mathcal{L} = \sum_{i}^{N}(-w_i\text{sim}(\mathbf{z}_i, \tilde{\mathbf{z}}_i) + \lambda \sum_{j,j\neq i}^{N} \text{sim}(\mathbf{z}_i, \tilde{\mathbf{z}}_j))\,, \tag{4.13}$$

where the second term is a regularization term employed to prevent the node embeddings from being collapsed into the same and λ is a trade-off hyperparameter.

Note that we only focus on maximizing the latent attribute predictability of high-score nodes without minimizing the predictability of low-score nodes in the above objective. These low-score nodes could also be normal nodes with high probability as the score from Eq.(4.12) is only obtained from the pre-trained model, resulting in the score not being fully reliable. If the predictability is also minimized for these nodes, conflicts would be induced for neighborhood prompt learning, limiting the performance of unsupervised GAD. After optimization, the latent attribute predictability is also directly used as the normal score for the final unsupervised GAD.

4.4 Experiments

4.4.1 Performance on Zero-shot Generalist GAD

Datasets. We evaluate UNPrompt on seven real-world GAD datasets from diverse social networks, online shopping co-review networks, and co-purchase networks. Specifically, the social networks include Facebook [114], Reddit [115] and Weibo [115]. The co-review networks consist of Amazon [95], YelpChi [96], Amazon-all and YelpChi-all. Disney [116] is a co-purchase network. The statistical information, including the number of nodes, edges, the dimension of the feature, and the anomaly rate of the datasets, can be found in Table 4.1.

Baseline Methods. Since there is no zero-shot generalist GAD method, a set of state-of-the-art (SotA) unsupervised and supervised baseline methods are employed for comparison. The unsupervised methods comprise reconstruction-based AnomalyDAE [29], contrastive learning-based CoLA [30], hop prediction-based HCM-A [100], local affinity-based TAM [32] and GADAM [117]. Supervised methods include two conventional GNNs – GCN [98] and GAT [21] – and three SotA GAD GNNs – BWGNN [35], GHRN [36] and XGBGraph [118]. As a graph prompting

Table 4.1 : Key statistics of the real-world GAD datasets with real anomalies.

Data set	Type	Nodes	Edges	Attributes	Anomalies(Rate)
Facebook	Social Networks	1,081	55,104	576	25(2.31%)
Reddit	Social Networks	10,984	168,016	64	366(3.33%)
Weibo	Social Networks	8,405	407,963	400	868(10.30%)
Amazon	Co-review	10,244	175,608	25	693(6.66%)
YelpChi	Co-review	24,741	49,315	32	1,217(4.91%)
Amazon-all	Co-review	11,944	4,398,392	25	821(6.87%)
YelpChi-all	Co-review	45,941	3,846,979	32	6,674(14.52%)
Disney	Co-purchase	124	335	28	6(4.84%)

method, GraphPrompt [105] is also used. Following [9, 32, 52], two widely-used metrics, AUROC and AUPRC, are used to evaluate the performance of all methods. For both metrics, the higher value denotes the better performance. Moreover, for each method, we report the average performance with standard deviations after 5 independent runs with different random seeds.

Implementation Details. To ensure a fair comparison, the common dimensionality is set to eight to unify the node attributes across graphs for all methods, and SVD is used for feature projection. The number of layers in GNNs is set to one and the number of hidden units is 128. The transformation layer is also implemented as a one-layer MLP with the same number of hidden units. The size of the neighborhood prompt is set to one. For all baselines, their recommended optimization and hyperparameter settings are used. Note that GraphPrompt is a graph prompting method. To adapt it for generalist GAD, after pre-training, we further perform the prompt learning in the source graph for supervised GAD. Then, the pre-trained model and learned prompts are used for anomaly detection on other datasets. UNPrompt and all baseline methods are trained on Facebook and then directly tested on the other GAD datasets. Facebook is used in the training since its anomaly patterns are more generic.

For the graph contrastive learning-based pre-training, the probabilities of edge removal and attribute masking are by default set to 0.2 and 0.3, respectively. Besides, the learning rate is set to 0.001 with the Adam optimizer, the training epoch is set to 200, and the temperature τ is 0.5. For the neighborhood prompt learning, the learning rate is also set to 0.001 with the Adam optimizer, and the training epoch is set to 900. Note that, since we focus on generalist GAD, we do not perform any hyperparameter search for specific target graphs. Instead, the results of all target graphs are obtained with the same hyperparameter settings.

For both generalist and unsupervised GAD, the code is implemented with Pytorch (version: 1.13.1), DGL (version: 1.0.1), OGB (version: 1.3.6), and Python 3.8.19. All experiments are conducted on a Linux server with an Intel CPU (Intel Xeon Gold 6346 3.1GHz) and an Nvidia A40 GPU.

Main Results. The results of all methods are presented in Table 4.2 and we can have the following observations. (1) Under the proposed generalist GAD scenario, where a model is trained on a single dataset and evaluated on seven other datasets, all the baseline baselines fail to work well, demonstrating that it is very challenging to build a generalist GAD model that generalizes across different datasets under the zero-shot setting. (2) For supervised methods, the simple GCN achieves better performance than the specially designed GAD GNNs. This can be attributed to more dataset-specific knowledge being captured in these specialized GAD models, limiting their generalization capacity to the unseen testing graphs. (3) Unsupervised methods perform more stably than supervised methods across the target graphs. This is because the unsupervised objectives are closer to the shared anomaly patterns across graphs compared to the supervised ones, especially for TAM, which employs a fairly generalized local affinity-based objective for GAD. (4) Despite being a graph prompting method, GraphPrompt fails to achieve promising performance for zero-

Table 4.2 : AUROC and AUPRC results on seven real-world GAD datasets with the models trained on Facebook only. For each dataset, the best performance per column within each metric is boldfaced, with the second-best underlined. "Avg" denotes the average performance.

Metric	Method	Dataset							
		Amazon	Reddit	Weibo	YelpChi	Aamzon-all	YelpChi-all	Disney	Avg.
AUROC	Unsupervised Methods								
	AnomalyDAE (ICASSP'20)	$0.5818_{\pm 0.039}$	$0.5016_{\pm 0.032}$	$\underline{0.7785}_{\pm 0.058}$	$0.4837_{\pm 0.094}$	$0.7228_{\pm 0.023}$	$0.5002_{\pm 0.018}$	$0.4853_{\pm 0.003}$	$\underline{0.5791}$
	CoLA (TNNLS'21)	$0.4580_{\pm 0.054}$	$0.4623_{\pm 0.005}$	$0.3924_{\pm 0.041}$	$0.4907_{\pm 0.017}$	$0.4091_{\pm 0.052}$	$0.4879_{\pm 0.010}$	$0.4696_{\pm 0.065}$	0.4529
	HCM-A (ECML-PKDD'22)	$0.4784_{\pm 0.005}$	$0.5387_{\pm 0.041}$	$0.5782_{\pm 0.048}$	$0.5000_{\pm 0.000}$	$0.5056_{\pm 0.059}$	$0.5023_{\pm 0.005}$	$0.2014_{\pm 0.015}$	0.4721
	TAM (NeurIPS'23)	$0.4720_{\pm 0.005}$	$\mathbf{0.5725}_{\pm 0.004}$	$0.4867_{\pm 0.028}$	$0.5035_{\pm 0.014}$	$0.7543_{\pm 0.002}$	$0.4216_{\pm 0.002}$	$0.4773_{\pm 0.003}$	0.5268
	GADAM (ICLR'24)	$\underline{0.6646}_{\pm 0.063}$	$0.4532_{\pm 0.024}$	$0.3652_{\pm 0.052}$	$0.3376_{\pm 0.012}$	$0.5959_{\pm 0.080}$	$0.4829_{\pm 0.016}$	$0.4288_{\pm 0.023}$	0.4755
	Supervised Methods								
	GCN (ICLR'17)	$0.5988_{\pm 0.016}$	$\underline{0.5645}_{\pm 0.000}$	$0.2232_{\pm 0.074}$	$0.5366_{\pm 0.019}$	$0.7195_{\pm 0.002}$	$\underline{0.5486}_{\pm 0.001}$	$0.5000_{\pm 0.000}$	0.5273
	GAT (ICLR'18)	$0.4981_{\pm 0.008}$	$0.5000_{\pm 0.025}$	$0.4521_{\pm 0.101}$	$\underline{0.5871}_{\pm 0.016}$	$0.5005_{\pm 0.012}$	$0.4802_{\pm 0.004}$	$0.5175_{\pm 0.054}$	0.5051
	BWGNN (ICML'22)	$0.4769_{\pm 0.020}$	$0.5208_{\pm 0.016}$	$0.4815_{\pm 0.108}$	$0.5538_{\pm 0.027}$	$0.3648_{\pm 0.050}$	$0.5282_{\pm 0.015}$	$0.6073_{\pm 0.026}$	0.5048
	GHRN (WebConf'23)	$0.4560_{\pm 0.033}$	$0.5253_{\pm 0.006}$	$0.5318_{\pm 0.038}$	$0.5524_{\pm 0.020}$	$0.3382_{\pm 0.085}$	$0.5125_{\pm 0.016}$	$0.5336_{\pm 0.030}$	0.4928
	XGBGraph (NeurIPS'23)	$0.4179_{\pm 0.000}$	$0.4601_{\pm 0.000}$	$0.5373_{\pm 0.000}$	$0.5722_{\pm 0.000}$	$\underline{0.7950}_{\pm 0.000}$	$0.4945_{\pm 0.000}$	$\mathbf{0.6692}_{\pm 0.000}$	0.5637
	GraphPrompt (WebConf'23)	$0.4904_{\pm 0.001}$	$0.4677_{\pm 0.001}$	$0.6135_{\pm 0.008}$	$0.3935_{\pm 0.002}$	$0.3215_{\pm 0.001}$	$0.4976_{\pm 0.000}$	$0.5192_{\pm 0.002}$	0.4719
	UNPrompt (Ours)	$\mathbf{0.7525}_{\pm 0.016}$	$0.5337_{\pm 0.002}$	$\mathbf{0.8860}_{\pm 0.007}$	$\mathbf{0.5875}_{\pm 0.016}$	$\mathbf{0.7962}_{\pm 0.022}$	$\mathbf{0.5558}_{\pm 0.012}$	$\underline{0.6412}_{\pm 0.030}$	**0.6790**
AUPRC	Unsupervised Methods								
	AnomalyDAE (ICASSP'20)	$0.0833_{\pm 0.015}$	$0.0327_{\pm 0.004}$	$\underline{0.6064}_{\pm 0.031}$	$0.0624_{\pm 0.017}$	$0.1921_{\pm 0.026}$	$0.1484_{\pm 0.009}$	$0.0566_{\pm 0.000}$	$\underline{0.1688}$
	CoLA (TNNLS'21)	$0.0669_{\pm 0.002}$	$0.0391_{\pm 0.004}$	$0.1189_{\pm 0.014}$	$0.0511_{\pm 0.000}$	$0.0861_{\pm 0.019}$	$0.1466_{\pm 0.003}$	$0.0701_{\pm 0.023}$	0.0827
	HCM-A (ECML-PKDD'22)	$0.0669_{\pm 0.002}$	$0.0391_{\pm 0.004}$	$0.1189_{\pm 0.014}$	$0.0511_{\pm 0.000}$	$0.0861_{\pm 0.019}$	$0.1466_{\pm 0.003}$	$0.0355_{\pm 0.001}$	0.0777
	TAM (NeurIPS'23)	$0.0666_{\pm 0.001}$	$\underline{0.0413}_{\pm 0.001}$	$0.1240_{\pm 0.014}$	$0.0524_{\pm 0.002}$	$0.1736_{\pm 0.004}$	$0.1240_{\pm 0.001}$	$0.0628_{\pm 0.001}$	0.0921
	GADAM (ICLR'24)	$0.1562_{\pm 0.103}$	$0.0293_{\pm 0.001}$	$0.0830_{\pm 0.005}$	$0.0352_{\pm 0.001}$	$0.1595_{\pm 0.121}$	$0.1371_{\pm 0.006}$	$0.0651_{\pm 0.012}$	0.0951
	Supervised Methods								
	GCN (ICLR'17)	$\underline{0.0891}_{\pm 0.007}$	$\mathbf{0.0439}_{\pm 0.000}$	$0.1109_{\pm 0.020}$	$0.0648_{\pm 0.009}$	$0.1536_{\pm 0.002}$	$\underline{0.1735}_{\pm 0.000}$	$0.0484_{\pm 0.000}$	0.0977
	GAT (ICLR'18)	$0.0688_{\pm 0.002}$	$0.0329_{\pm 0.002}$	$0.1009_{\pm 0.017}$	$\mathbf{0.0810}_{\pm 0.005}$	$0.0696_{\pm 0.001}$	$0.1400_{\pm 0.002}$	$0.0530_{\pm 0.004}$	0.0780
	BWGNN (ICML'22)	$0.0652_{\pm 0.002}$	$0.0389_{\pm 0.003}$	$0.2241_{\pm 0.046}$	$0.0708_{\pm 0.018}$	$0.0586_{\pm 0.003}$	$0.1605_{\pm 0.005}$	$0.0624_{\pm 0.003}$	0.0972
	GHRN (WebConf'23)	$0.0633_{\pm 0.003}$	$0.0407_{\pm 0.002}$	$0.1965_{\pm 0.059}$	$0.0661_{\pm 0.010}$	$0.0569_{\pm 0.006}$	$0.1505_{\pm 0.005}$	$0.0519_{\pm 0.003}$	0.0894
	XGBGraph (NeurIPS'23)	$0.0536_{\pm 0.000}$	$0.0330_{\pm 0.000}$	$0.2256_{\pm 0.000}$	$0.0655_{\pm 0.000}$	$\underline{0.2307}_{\pm 0.000}$	$0.1449_{\pm 0.000}$	$\underline{0.1215}_{\pm 0.000}$	0.1250
	GraphPrompt (WebConf'23)	$0.0661_{\pm 0.000}$	$0.0334_{\pm 0.000}$	$0.2014_{\pm 0.004}$	$0.0382_{\pm 0.000}$	$0.0666_{\pm 0.000}$	$0.1542_{\pm 0.000}$	$0.0617_{\pm 0.001}$	0.0888
	UNPrompt (Ours)	$\mathbf{0.1602}_{\pm 0.013}$	$0.0351_{\pm 0.000}$	$\mathbf{0.6406}_{\pm 0.026}$	$\underline{0.0712}_{\pm 0.008}$	$\mathbf{0.2430}_{\pm 0.028}$	$\mathbf{0.1810}_{\pm 0.012}$	$\mathbf{0.1236}_{\pm 0.031}$	**0.2078**

shot generalist GAD. This is because there is no label information provided, and further model optimization during inference. (5) The proposed method UNPrompt demonstrates strong and stable generalist GAD capacity across graphs from different distributions and domains. Specifically, UNPrompt achieves the best AUROC performance on 5 out of 7 datasets and the average performance outperforms the best baseline method by over 9%. The superiority is attributed to the fact that i) the

proposed coordinate-wise normalization effectively aligns the features across graphs, and ii) the shared generalized normal and abnormal patterns are well captured in the neighborhood prompts.

Graph Similarity In addition to the visualization results presented in Figure 4.2, we further provide the distributional similarity of various graphs in this section. Specifically, for dimension-aligned graphs across different distributions and domains, we measure their distributional similarity to analyze their diverse semantics.

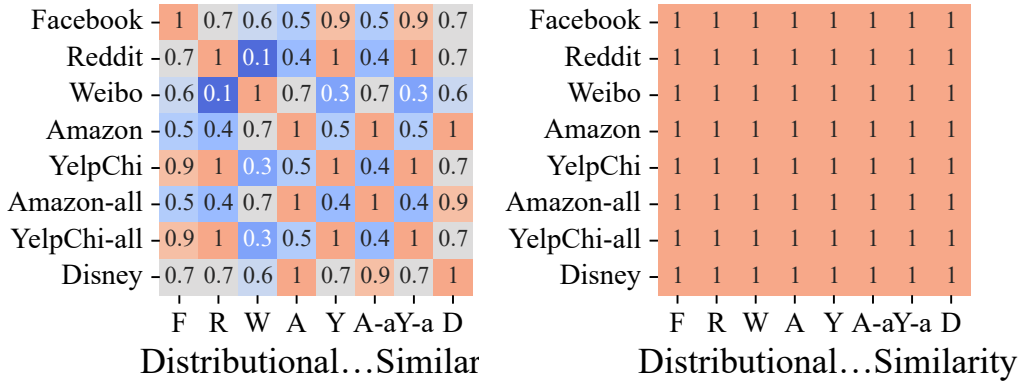
Given a graph $\mathcal{G}^{(i)} = (A^{(i)}, \tilde{X}^{(i)})$, the coordinate-wise mean $\boldsymbol{\mu}^{(i)} = [\mu_1^{(i)}, \ldots, \mu_{d'}^{(i)}]$ and variance $\boldsymbol{\sigma}^{(i)} = [\sigma_1^{(i)}, \ldots, \sigma_{d'}^{(i)}]$ of $\tilde{X}^{(i)}$ are calculated and concatenated to form the distributional vector of $\mathcal{G}^{(i)}$, *i.e.*, $\mathbf{d}_i = [\boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{(i)}]$. Then, the distribution similarity between $\mathcal{G}^{(i)}$ and $\mathcal{G}^{(j)}$ is measured via the cosine similarity,

$$s_{ij} = \mathrm{sim}(\mathbf{d}_i, \mathbf{d}_j)\,. \tag{4.14}$$

The distributional similarities between graphs from different domains or distributions are shown in Figure 4.3(a).. From the figure, we can see that the distributional similarities are typically small, demonstrating the diverse semantics of node features across graphs. Noth that, for Amazon & Amazon-all and YelpChi & YelpChi-all, their distribution similarity is one, which can be attributed to the fact that they are from the same distributions respectively but with different numbers of nodes and structures.

To reduce the semantic gap among graphs for generalist GAD, we propose to calibrate the distributions of all graphs into the same frame with coordinate-wise normalization. The distributional similarity with normalization is illustrated in Figure 4.3(b). It is clear that the node attributes share the same distribution after the normalization. In this way, the generalist model can better capture the shared GAD patterns and generalize to different target graphs, as demonstrated in our

experimental results.



(a) (b)

Figure 4.3 : **(a)** Distributional similarity between different graphs without coordinate-wise normalization. **(b)** Distributional similarity between different graphs with the coordinate-wise normalization.

Ablation Study. To evaluate the importance of each component in UNPrompt, we design four variants, *i.e.*, w/o coordinate-wise normalization, w/o graph contrastive learning-based pre-training, without neighborhood prompts, and w/o transformation layer. The results of these variants are reported in the Table 4.3. Due to the page limits, four datasets are used, including Amazon, Reddit, Weibo and YelpChi. From the table, we can see that all four components contribute to the overall superior performance of UNPrompt. More specifically, (1) without the coordinate-wise normalization, the method fails to calibrate the distributions of diverse node attributes into a common space, leading to a large performance drop across all datasets. (2) Besides the semantics alignment, the graph contrastive learning-based pre-training ensures our GNN network is transferable to other graphs instead of overfitting to the training graph. As expected, the performance of the variant without pre-training also drops significantly. (3) If the neighborhood prompts are

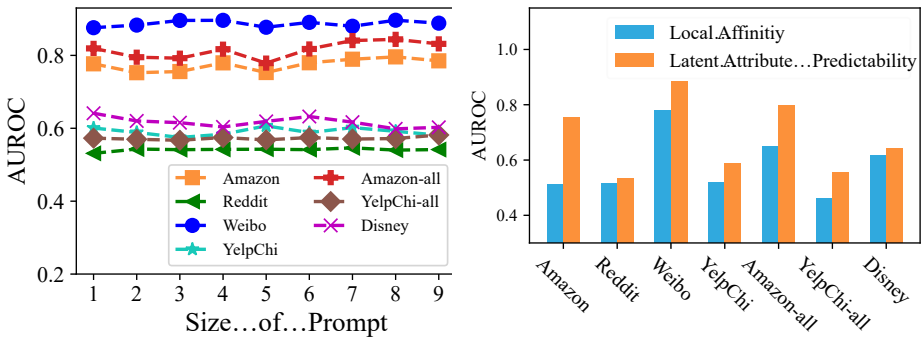
removed, the learning of latent node attribute prediction is ineffective for capturing generalized normal and abnormal patterns. (4) The variant without the transformation layer achieves inferior performance on nearly all the datasets, demonstrating the importance of mapping the features into a more anomaly-discriminative space.

Table 4.3 : AUROC results of UNPrompt and its four variants.

Method	Amazon	Reddit	Weibo	YelpChi	Avg.
UNPrompt	**0.7525**	0.5337	**0.8860**	**0.5875**	**0.6899**
w/o Normalization	0.4684	0.5006	0.1889	0.5620	0.4300
w/o Pre-training	0.5400	0.5233	0.5658	0.4672	0.5241
w/o Prompt	0.5328	0.5500	0.4000	0.4520	0.4837
w/o Transformation	0.7331	**0.5556**	0.7406	0.5712	0.6501

Sensitivity w.r.t the Neighborhood Prompt Size. We evaluate the sensitivity of UNPrompt w.r.t the size of the neighborhood prompts *i.e.*, K. We vary K in the range of $[1, 9]$ and report the results in Figure 4.4(a). It is clear that the performances on Reddit, Weibo and YelpChi-all remain stable with varying sizes of neighborhood prompts, while the other datasets show slight fluctuation, demonstrating that the generalized normal and abnormal patterns can be effectively captured in our neighborhood prompts even with a small size.

Prompt learning using latent attribute prediction vs. alternative graph anomaly measures. To further justify the effectiveness of latent attribute predictability on learning generalized GAD patterns in our prompt learning module, we compare our learnable anomaly measure to the recently proposed anomaly measure, local node affinity [32]. All modules of UNPrompt are fixed with only the latent attribute prediction task replaced as the maximization of local affinity as in TAM. The results are presented in Figure 4.4(b). We can see that the latent attribute predictability consistently and significantly outperforms the local affinity-based measure



(a) (b)

Figure 4.4 : (**a**) AUROC results of UNPrompt w.r.t. varying neighborhood prompt size. (**b**). The AUROC performance of generalist GAD with different prompt learning objectives.

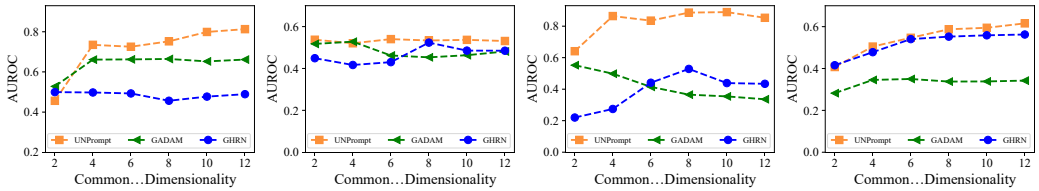
across all graphs, demonstrating its superiority in learning generalized patterns for generalist GAD.

4.4.2 Generalist Performance with different common dimensionalities

For the results reported above, the common dimensionality is set to eight. In this subsection, we further evaluate the generalist anomaly detection with different common dimensionalities. Specifically, the dimensionality varies in $[2, 4, 6, 8, 10, 12]$ and the results are reported in Figure 4.5. Without loss of generality, we conduct the analysis on Amazon, Reddit, Weibo and YelpChi and employ two baselines (GADAM and GHRN) for comparisons which are unsupervised and supervised methods, respectively.

From the figure, we can see that small dimensionality leads to poor generalist anomaly detection performance of UNPrompt. This is attributed to the fact that much attribute information would be discarded with a small dimensionality. By increasing the common dimensionality, more attribute information is retained, generally resulting in much better detection performance. However, this trend does

not hold for the baselines as they achieve inconsistent performance gain or loss on different datasets with the increase of the common dimensionality. This is largely attributed to the fact that these one-model-for-one-dataset methods fail to capture the generalized anomaly measure.



(a) Amazon (b) Reddit (c) Weibo (d) YelpChi

Figure 4.5 : AUROC results of UNPrompt and other baselines with different common dimensionalities.

4.4.3 Generalist performance with different training graphs

In the above, we report the generalist performance of UNPrompt by using Facebook as the training graph. To further demonstrate the generalizability of UNPrompt, we conduct additional experiments by using Amazon as the training graph and testing the learned generalist model on the rest graphs. Note that Facebook and Amazon are from different domains, which are the social network and the co-review network, respectively.

The AUROC and AUPRC results of all methods are reported in Table 4.4. Similar to the observations when taking Facebook as the training graph, UNPrompt achieves the best average performance in terms of both AUROC and AUPRC when training on Amazon, demonstrating the generalizability and effectiveness of UNPrompt with different training graphs. Note that the training graph Amazon and the target graph Amazon-all come from the same distribution but have different numbers of nodes and graph structures. Intuitively, all the methods should achieve

Table 4.4 : AUROC and AUPRC results on seven real-world GAD datasets with the generalist model trained on Amazon. For each dataset and metric, the best performance per column is boldfaced, with the second-best underlined. "Avg" denotes the average performance of each method.

Metric	Method	Dataset							
		Facebook	Reddit	Weibo	YelpChi	Aamzon-all	YelpChi-all	Disney	Avg.
AUROC	Unsupervised Methods								
	AnomalyDAE	$0.6123_{\pm 0.141}$	$\mathbf{0.5799}_{\pm 0.035}$	$\underline{0.7884}_{\pm 0.031}$	$0.4788_{\pm 0.046}$	$0.6233_{\pm 0.070}$	$0.4912_{\pm 0.009}$	$0.4938_{\pm 0.005}$	0.5811
	CoLA	$0.5427_{\pm 0.109}$	$0.4962_{\pm 0.025}$	$0.3987_{\pm 0.017}$	$0.3358_{\pm 0.012}$	$0.4751_{\pm 0.014}$	$0.4937_{\pm 0.003}$	$0.5455_{\pm 0.031}$	0.4697
	HCM-A	$0.5044_{\pm 0.047}$	$0.4993_{\pm 0.057}$	$0.4937_{\pm 0.056}$	$0.5000_{\pm 0.000}$	$0.4785_{\pm 0.016}$	$0.4958_{\pm 0.003}$	$0.2051_{\pm 0.034}$	0.4538
	TAM	$0.5496_{\pm 0.038}$	$\underline{0.5764}_{\pm 0.003}$	$0.4876_{\pm 0.029}$	$0.5091_{\pm 0.014}$	$0.7525_{\pm 0.002}$	$0.4268_{\pm 0.002}$	$0.4850_{\pm 0.004}$	0.5410
	GADAM	$0.6024_{\pm 0.033}$	$0.4720_{\pm 0.062}$	$0.4324_{\pm 0.047}$	$0.4299_{\pm 0.023}$	$0.5199_{\pm 0.072}$	$0.5289_{\pm 0.017}$	$0.3966_{\pm 0.021}$	0.4832
	Supervised Methods								
	GCN	$\underline{0.6892}_{\pm 0.004}$	$0.5658_{\pm 0.000}$	$0.2355_{\pm 0.019}$	$0.5277_{\pm 0.002}$	$0.7503_{\pm 0.002}$	$0.5565_{\pm 0.000}$	$0.5000_{\pm 0.000}$	0.5464
	GAT	$0.3886_{\pm 0.118}$	$0.4997_{\pm 0.012}$	$0.3897_{\pm 0.134}$	$0.5051_{\pm 0.019}$	$0.5007_{\pm 0.006}$	$0.4977_{\pm 0.006}$	$0.5840_{\pm 0.111}$	0.4808
	BWGNN	$0.5441_{\pm 0.020}$	$0.4026_{\pm 0.028}$	$0.4214_{\pm 0.039}$	$0.4908_{\pm 0.013}$	$\underline{0.9684}_{\pm 0.005}$	$0.5841_{\pm 0.062}$	$0.4196_{\pm 0.047}$	0.5473
	GHRN	$0.5242_{\pm 0.013}$	$0.4096_{\pm 0.021}$	$0.4783_{\pm 0.021}$	$0.5036_{\pm 0.016}$	$0.9601_{\pm 0.018}$	$\mathbf{0.6045}_{\pm 0.022}$	$0.4000_{\pm 0.098}$	0.5543
	XGBGraph	$0.4869_{\pm 0.069}$	$0.4869_{\pm 0.069}$	$0.7843_{\pm 0.090}$	$0.4773_{\pm 0.022}$	$\mathbf{0.9815}_{\pm 0.000}$	$\underline{0.5869}_{\pm 0.014}$	$0.4376_{\pm 0.044}$	$\underline{0.6059}$
	GraphPrompt	$0.3093_{\pm 0.000}$	$0.4511_{\pm 0.000}$	$0.2032_{\pm 0.000}$	$\mathbf{0.7093}_{\pm 0.000}$	$0.6331_{\pm 0.000}$	$0.4994_{\pm 0.000}$	$\mathbf{0.7128}_{\pm 0.000}$	0.5026
	Our	$\mathbf{0.7917}_{\pm 0.021}$	$0.5356_{\pm 0.005}$	$\mathbf{0.8192}_{\pm 0.015}$	$\underline{0.5362}_{\pm 0.007}$	$0.9289_{\pm 0.007}$	$0.5448_{\pm 0.009}$	$\underline{0.6959}_{\pm 0.042}$	**0.6932**
AUPRC	Unsupervised Methods								
	AnomalyDAE	$\underline{0.0675}_{\pm 0.028}$	$0.0413_{\pm 0.005}$	$\mathbf{0.6172}_{\pm 0.015}$	$0.0647_{\pm 0.016}$	$0.1025_{\pm 0.026}$	$0.1479_{\pm 0.006}$	$0.0583_{\pm 0.001}$	0.1571
	CoLA	$0.0468_{\pm 0.026}$	$0.0327_{\pm 0.002}$	$0.0956_{\pm 0.005}$	$0.0361_{\pm 0.001}$	$0.0678_{\pm 0.005}$	$0.1474_{\pm 0.001}$	$0.0717_{\pm 0.015}$	0.0712
	HCM-A	$0.0249_{\pm 0.003}$	$0.0374_{\pm 0.008}$	$0.0979_{\pm 0.011}$	$0.0511_{\pm 0.000}$	$0.0727_{\pm 0.006}$	$0.1453_{\pm 0.000}$	$0.0452_{\pm 0.020}$	0.0678
	TAM	$0.0243_{\pm 0.002}$	$\underline{0.0417}_{\pm 0.001}$	$0.1266_{\pm 0.015}$	$0.0532_{\pm 0.002}$	$0.1771_{\pm 0.002}$	$0.1271_{\pm 0.001}$	$0.0682_{\pm 0.002}$	0.0883
	GADAM	$0.0461_{\pm 0.014}$	$0.0299_{\pm 0.004}$	$0.0917_{\pm 0.007}$	$0.0428_{\pm 0.002}$	$0.0773_{\pm 0.024}$	$0.1602_{\pm 0.010}$	$0.0732_{\pm 0.004}$	0.0745
	Supervised Methods								
	GCN	$0.0437_{\pm 0.001}$	$\mathbf{0.0449}_{\pm 0.000}$	$0.2527_{\pm 0.026}$	$\underline{0.0763}_{\pm 0.001}$	$0.1738_{\pm 0.002}$	$0.1759_{\pm 0.000}$	$0.0484_{\pm 0.000}$	0.1165
	GAT	$0.0445_{\pm 0.039}$	$0.0327_{\pm 0.001}$	$0.0892_{\pm 0.016}$	$0.0595_{\pm 0.003}$	$0.0697_{\pm 0.001}$	$0.1478_{\pm 0.003}$	$0.0760_{\pm 0.035}$	0.0742
	BWGNN	$0.0289_{\pm 0.003}$	$0.0263_{\pm 0.002}$	$0.2735_{\pm 0.026}$	$0.0543_{\pm 0.004}$	$\underline{0.8406}_{\pm 0.012}$	$0.1975_{\pm 0.031}$	$0.0494_{\pm 0.004}$	0.2101
	GHRN	$0.0254_{\pm 0.001}$	$0.0265_{\pm 0.002}$	$0.3103_{\pm 0.013}$	$0.0541_{\pm 0.005}$	$0.8142_{\pm 0.045}$	$\mathbf{0.2015}_{\pm 0.015}$	$0.0561_{\pm 0.028}$	0.2126
	XGBGraph	$0.0268_{\pm 0.006}$	$0.0315_{\pm 0.000}$	$0.4116_{\pm 0.040}$	$0.0500_{\pm 0.003}$	$\mathbf{0.8673}_{\pm 0.000}$	$\underline{0.1994}_{\pm 0.012}$	$0.0541_{\pm 0.005}$	$\underline{0.2344}$
	GraphPrompt	$0.0169_{\pm 0.000}$	$0.0298_{\pm 0.000}$	$0.0812_{\pm 0.000}$	$\mathbf{0.0975}_{\pm 0.000}$	$0.1368_{\pm 0.000}$	$0.1481_{\pm 0.000}$	$\mathbf{0.1157}_{\pm 0.000}$	0.0894
	Our	$\mathbf{0.2291}_{\pm 0.023}$	$0.0340_{\pm 0.001}$	$\underline{0.4746}_{\pm 0.033}$	$0.0610_{\pm 0.003}$	$0.7329_{\pm 0.042}$	$0.1767_{\pm 0.004}$	$\underline{0.0933}_{\pm 0.013}$	**0.2574**

promising performance on Amazon-all. However, only a few methods achieve this goal, including BWGNN, GHRN, XGBGraph, and our method. The failures of other baselines can be attributed to the more complex graph structure of Amazon-all hinders the generalizability of these methods. Moreover, compared to BWGNN, GHRN

and XGBGraph, our method performs more stably across different datasets. This demonstrates the importance of capturing intrinsic normal and abnormal patterns for graph anomaly detection.

Table 4.5 : Training time and inference time (seconds) for different methods.

Methods	AnomalyDAE	TAM	GAT	BWGNN	UNPrompt
Training Time	86.04	479.70	2.43	4.86	2.08
Inference Time	264.29	521.92	300.90	330.99	58.95

Time Complexity Analysis. In Table 4.5, we report the training time and inference time of different methods, where two representative unsupervised methods (AnomalyDAE and TAM) and two supervised methods (GAT and BWGNN) are used for comparison to UNPrompt. The results show that the proposed method requires much less training and inference time compared to other baselines, demonstrating the efficiency of the proposed UNPrompt. Note that TAM has the highest time consumption, which can be attributed to that it performs multiple graph truncations and learns multiple local affinity maximization networks.

4.4.4 Performance in Unsupervised GAD

Experimental Setup.

Six datasets from different distributions and domains are used, *i.e.*, Amazon, Facebook, Reddit, YelpChi, Amazon-all, and YelpChi-all. Following [32], eight SotA unsupervised baselines are used for comparison, *i.e.*, iForest [119], ANOMALOUS [99], CoLA [30], SL-GAD [31], HCM-A [100], DOMINANT [28], ComGA [120] and TAM [32]. For each method, we report the average performance with standard deviations after 5 independent runs with different random seeds. The implementa-

tion details of UNPrompt remain the same as in the generalist GAD setting. Specifically, the hidden units of the neighborhood aggregation network and the transformation layer are set to 128 for all graphs. The threshold t is determined by the 40th percentile of the normal scores obtained by the pre-trained model g, and the scaling parameter α is set to 10 for all graphs. Besides, we utilize random search to find the optimal hyperparameters of the size of neighborhood prompts K and the trade-off parameter λ.

Main Results.

The AUROC and AUPRC results of all methods are presented in Table 4.6. Despite being a generalist GAD method, UNPrompt works very well as a specialized GAD model too. UNPrompt substantially outperforms all the baseline methods on all datasets in terms of both AUROC and AUPRC. Particularly, the average performance of UNPrompt surpasses the best baseline method TAM by over 2% in both metrics. Moreover, UNPrompt outperforms the best baseline method by 2%-6% in AUROC on most of the datasets. The superior performance shows that the latent node attribute predictability can be a generalized GAD measure that holds for different graphs, and this property can be effectively learned by the proposed neighborhood prompting method.

Sensitivity of Unsuperivised GAD w.r.t the threshold To evaluate the sensitivity of the unsupervised GAD performance of our method w.r.t the threshold, we vary the threshold in the range of $[5\%, 50\%]$ at a step size of 5%. Without loss of generality, three datasets are employed, including Facebook, Reddit, and Amazon. The results of these datasets are reported in Table 4.7.

From the table, we can see that the unsupervised performance of UNPrompt can perform well and stably with a sufficiently large threshold (e.g., no less than 30%),

Table 4.6 : AUROC and AUPRC results of unsupervised GAD methods on six real-world GAD datasets. The best performance per column within each metric is boldfaced, with the second-best underlined. "Avg" denotes the average performance of each method.

Metric	Method	Dataset						
		Amazon	Facebook	Reddit	YelpChi	Amazon-all	YelpChi-all	Avg.
AUROC	iForest (TKDD'12)	$0.5621_{\pm 0.008}$	$0.5382_{\pm 0.015}$	$0.4363_{\pm 0.020}$	$0.4120_{\pm 0.040}$	$0.1914_{\pm 0.002}$	$0.3617_{\pm 0.001}$	0.4169
	ANOMALOUS (IJCAI'18)	$0.4457_{\pm 0.003}$	$0.9021_{\pm 0.005}$	$0.5387_{\pm 0.012}$	$0.4956_{\pm 0.003}$	$0.3230_{\pm 0.021}$	$0.3474_{\pm 0.018}$	0.5087
	DOMINANT (SIAM'19)	$0.5996_{\pm 0.004}$	$0.5677_{\pm 0.002}$	$0.5555_{\pm 0.011}$	$0.4133_{\pm 0.010}$	$0.6937_{\pm 0.028}$	$0.5390_{\pm 0.014}$	0.5615
	CoLA (TNNLS'21)	$0.5898_{\pm 0.008}$	$0.8434_{\pm 0.011}$	$\underline{0.6028}_{\pm 0.007}$	$0.4636_{\pm 0.001}$	$0.2614_{\pm 0.021}$	$0.4801_{\pm 0.016}$	0.5402
	SL-GAD (TKDE'21)	$0.5937_{\pm 0.011}$	$0.7936_{\pm 0.005}$	$0.5677_{\pm 0.005}$	$0.3312_{\pm 0.035}$	$0.2728_{\pm 0.012}$	$0.5551_{\pm 0.015}$	0.5190
	HCM-A (ECML-PKDD'22)	$0.3956_{\pm 0.014}$	$0.7387_{\pm 0.032}$	$0.4593_{\pm 0.011}$	$0.4593_{\pm 0.005}$	$0.4191_{\pm 0.011}$	$0.5691_{\pm 0.018}$	0.5069
	ComGA (WSDM'22)	$0.5895_{\pm 0.008}$	$0.6055_{\pm 0.000}$	$0.5453_{\pm 0.003}$	$0.4391_{\pm 0.000}$	$0.7154_{\pm 0.014}$	$0.5352_{\pm 0.006}$	0.5716
	TAM (NeurIPS'23)	$\underline{0.7064}_{\pm 0.010}$	$\underline{0.9144}_{\pm 0.008}$	$0.6023_{\pm 0.004}$	$\underline{0.5643}_{\pm 0.007}$	$\underline{0.8476}_{\pm 0.028}$	$\underline{0.5818}_{\pm 0.033}$	$\underline{0.7028}$
	UNPrompt (Ours)	$\mathbf{0.7335}_{\pm 0.020}$	$\mathbf{0.9379}_{\pm 0.006}$	$\mathbf{0.6067}_{\pm 0.006}$	$\mathbf{0.6223}_{\pm 0.007}$	$\mathbf{0.8516}_{\pm 0.004}$	$\mathbf{0.6084}_{\pm 0.001}$	**0.7267**
AUPRC	iForest (TKDD'12)	$0.1371_{\pm 0.002}$	$0.0316_{\pm 0.003}$	$0.0269_{\pm 0.001}$	$0.0409_{\pm 0.000}$	$0.0399_{\pm 0.001}$	$0.1092_{\pm 0.001}$	0.0643
	ANOMALOUS (IJCAI'18)	$0.0558_{\pm 0.001}$	$0.1898_{\pm 0.004}$	$0.0375_{\pm 0.004}$	$0.0519_{\pm 0.002}$	$0.0321_{\pm 0.001}$	$0.0361_{\pm 0.005}$	0.0672
	DOMINANT (SIAM'19)	$0.1424_{\pm 0.002}$	$0.0314_{\pm 0.041}$	$0.0356_{\pm 0.002}$	$0.0395_{\pm 0.020}$	$0.1015_{\pm 0.018}$	$0.1638_{\pm 0.007}$	0.0857
	CoLA (TNNLS'21)	$0.0677_{\pm 0.001}$	$0.2106_{\pm 0.017}$	$\underline{0.0449}_{\pm 0.002}$	$0.0448_{\pm 0.002}$	$0.0516_{\pm 0.001}$	$0.1361_{\pm 0.015}$	0.0926
	SL-GAD (TKDE'21)	$0.0634_{\pm 0.005}$	$0.1316_{\pm 0.020}$	$0.0406_{\pm 0.004}$	$0.0350_{\pm 0.000}$	$0.0444_{\pm 0.001}$	$0.1711_{\pm 0.011}$	0.0810
	HCM-A (ECML-PKDD'22)	$0.0527_{\pm 0.015}$	$0.0713_{\pm 0.004}$	$0.0287_{\pm 0.005}$	$0.0287_{\pm 0.012}$	$0.0565_{\pm 0.003}$	$0.1154_{\pm 0.004}$	0.0589
	ComGA (WSDM'22)	$0.1153_{\pm 0.005}$	$0.0354_{\pm 0.001}$	$0.0374_{\pm 0.001}$	$0.0423_{\pm 0.000}$	$0.1854_{\pm 0.003}$	$0.1658_{\pm 0.003}$	0.0969
	TAM (NeurIPS'23)	$\underline{0.2634}_{\pm 0.008}$	$\underline{0.2233}_{\pm 0.016}$	$0.0446_{\pm 0.001}$	$\underline{0.0778}_{\pm 0.009}$	$\underline{0.4346}_{\pm 0.021}$	$\underline{0.1886}_{\pm 0.017}$	$\underline{0.2054}$
	UNPrompt (Ours)	$\mathbf{0.2688}_{\pm 0.060}$	$\mathbf{0.2622}_{\pm 0.028}$	$\mathbf{0.0450}_{\pm 0.001}$	$\mathbf{0.0895}_{\pm 0.004}$	$\mathbf{0.6094}_{\pm 0.014}$	$\mathbf{0.2068}_{\pm 0.004}$	**0.2470**

but it may drop significantly with a small threshold, e.g., thresholds like 5%-15% that are close to the ground-truth anomaly rate. This is because more abnormal nodes would have a substantially higher chance of being mistakenly treated as normal nodes with such a small threshold, which would in turn mislead the optimization and subsequently degrade the GAD performance. By contrast, increasing the threshold would lift the acceptance bar of normal nodes, allowing the optimization to focus on high-confident normal nodes and effectively mitigate the adverse effects caused by the wrongly labeled normal nodes.

Table 4.7 : AUROC results of UNPrompt with different thresholds (%) in unsupervised GAD.

Datasets	5	10	15	20	25	30	35	40	45	50
Facebook	0.8859	0.9125	0.9218	0.9231	0.9210	0.9272	0.9340	0.9379	0.9369	0.9255
Reddit	0.5667	0.5754	0.5802	0.5824	0.5942	0.5871	0.5864	0.6067	0.5906	0.5828
Amazon	0.6264	0.6960	0.7152	0.7393	0.7394	0.7463	0.7462	0.7335	0.7392	0.7389

4.5 Conclusion

In this chapter, we propose a novel zero-shot generalist GAD method, UNPrompt, that trains one detector on a single dataset and can effectively generalize to other unseen target graphs without any further re-training or labeled nodes of target graphs during inference. The attribute inconsistency and the absence of generalized anomaly patterns are the main obstacles for generalist GAD. To address these issues, two main modules are proposed, *i.e.*, coordinate-wise normalization-based attribute unification and neighborhood prompt learning. The first module aligns node attribute dimensionality and semantics, while the second module captures generalized normal and abnormal patterns via the neighborhood-based latent node attribute prediction. Extensive experiments on both generalist and unsupervised GAD demonstrate the effectiveness of UNPrompt.

Chapter 5

Graph Continual Learning with Debiased Lossless Memory Replay

5.1 Introduction

Due to the superior capacity to represent complex relations between samples, graph is widely used in various real-world applications [1–3] such as social networks, citation networks, and online shopping. For example, in the context of online shopping, consumers could be nodes and the edges between consumers could represent that they have purchased or rated the same product. In real-world applications, graph data are often expanded continually, *e.g.*, new consumers and the associated connections would be constantly added to the online shopping network. Following the message propagation paradigm, graph neural networks (GNNs) [6, 98, 112, 121] have achieved remarkable success for various graph-related tasks. Despite the success, most GNNs operate on static graph data. Directly applying them to accommodate new emerging graphs would cause the GNNs to easily forget the knowledge learned from the previous data due to the large distribution difference between the historical data and the newly added data. The forgetting of the learned knowledge when learning new graphs, a.k.a. catastrophic forgetting, would result in deteriorated performance on historical graphs. A simple solution is to store all historical data and repeatedly retrain the GNNs whenever the graph is updated, but it is prohibitively expensive in terms of computational time and resources, considering the large scale of the continually expanding graph. Moreover, the previous graph data would also be inaccessible in privacy-critical application scenarios when learning the

newly added data.

To tackle catastrophic forgetting, many methods have been proposed and demonstrated impressive performance on Euclidean data such as images and texts [122–126]. However, it is ineffective to directly adopt them for graph continual learning (GCL) [11,12,127] as graphs are non-Euclidean data that contain complex relations among a large number of nodes.

Recently, to address the unique challenges in graph data, several GCL works [53–63] have been proposed to continually adapt GNNs to the expanded graph data of the current task while maintaining the performance over the graph data of previous tasks. Generally, these methods can be roughly divided into three categories: regularization-based methods, parameter isolation-based methods, and replay-based methods. Due to the straightforward intuitiveness and impressive effectiveness, replay-based methods [54,61,63] have been widely explored and achieved remarkable capacity against catastrophic forgetting in GCL. Typically, they sample and store representative data of previous tasks in a memory buffer and replay them when learning a new task. However, since only selected graph information is stored, their constructed memory for each previous task often struggles to capture the global structure information of the full graph (*i.e.*, **holistic graph information**), limiting the power of memory replay against catastrophic forgetting. For example, ERGNN [54] stores only individual nodes via sampling methods for replaying, which ignores the rich relations among the nodes, severely degrading the effectiveness of memory replay in GCL. A straightforward solution to this issue is to store the complete neighbors and the associated edges of the selected nodes for graph data of previous tasks, but this would pose great challenges to memory storage and is infeasible under **tight memory budgets**. To preserve the topological information and alleviate the storage requirement simultaneously, recent studies propose approaches like SSM [63] to sparsify the neighborhood of the selected nodes by fil-

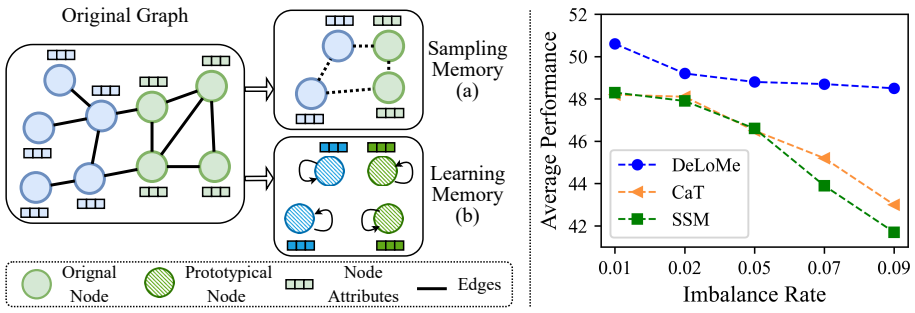


Figure 5.1 : ***Left:*** (a) Current replay-based methods use a sampling-based memory consisting of partially sampled graph data. (b) Our approach learns to generate the memory using a lossless small graph with prototypical node representations. ***Right:*** Average accuracy (AA) of three replay methods – SSM, CaT and our DeLoMe – with increasing imbalance rates on Arxiv.

tering unimportant ones. Nevertheless, all these methods focus on constructing the memory using partial graph data, as shown in Figure 5.1(Left), failing to preserve the holistic graph structure. Also, these memory construction methods would become inapplicable in privacy-critical applications where the replay of previous graph data can lead to **privacy leakage**, *e.g.*, storing the original data of online shopping networks would divulge the purchase/rating information of consumers.

Further, since the amount of the memory data is often limited, the current graph data often dominates the training data, leading to a **data imbalance** between the classes in the memory data and that in the current graph data. When updating the GCL models with those imbalanced data, the models are biased toward the current task, amplifying the deteriorated performance on the previous tasks as the graph continually expands with a fixed memory budget (*i.e.*, increasing imbalance rates), *e.g.*, performance of SSM in Figure 5.1(Right).

To tackle these three issues, in this chapter, we introduce a novel memory replay-based GCL approach, called Debiased Lossless Memory replay (**DeLoMe**). Rather

than storing the original graph data, it learns a small graph consisting of *prototypical node representations as memory* so that the gradient of a randomly initialized GNN on the original large graph is lossless compared to that on the learned small graph. In doing so, the learned prototypical representations are enforced to better capture the holistic graph structure and attribute information, resulting in better performance on previous graph data, as illustrated in Figure 5.1(Right). This node representation-based memory also helps better preserve the privacy of the graph data, compared to the original node/edge-based memory.

To handle the aforementioned bias issue, a debiased loss function is devised in our GCL objective. This is done by calibrating the prediction logits of the classes in the memory data and the current graph data in the continual updating of our DeLoMe model, which helps largely enhance its robustness w.r.t. the class imbalance.

5.2 Related Work

Graph Condensation Graph condensation aims to compress a graph into a significantly smaller graph while preserving the holistic information of the original graph. Various graph compression methods have been proposed [128–132] that are based on gradient matching or distribution matching between the compressed graph and the original graph. In this chapter, we utilize gradient matching to compress the graphs with node features and structure in previous tasks into comprehensive prototypical node representations, which can serve as the memory for subsequent replay. Note that the concurrent work CaT [64] shares a common motivation with our method. However, unlike CaT which adopts a distribution matching-based method to learn the memory, we propose to use a gradient matching method that can measure the loss of condensation in a more fine-grained manner. CaT performs the GNNs training within the learned memory bank to alleviate the class imbalance problem, but it can lead to less effective utilization of the graph data of the current

task. We instead devise a debiased GCL objective to calibrate the GCL predictions while avoiding the inefficient exploitation of the current graph data.

5.3 Preliminaries

5.3.1 The GCL Problem

In this chapter, we focus on the node-level GCL problem. Formally, this problem can be formulated as learning a model on a sequence of graphs (tasks) $\{\mathcal{G}_1, \ldots, \mathcal{G}_T\}$ where T is the number of continual learning tasks. Each $\mathcal{G}_t = (A_t, X_t)$ is a newly emerging graph at task t, where A_t denotes the relations between nodes, X_t represents the node features, and the labels of nodes can be denoted as Y_t. Generally, each task contains a unique set of classes, *i.e.*, $\{Y_t \cap Y_j = \varnothing | t \neq j\}$. When learning task t, the model trained from previous tasks only has access to current data $\mathcal{G}_t$. The goal is to adapt the model to current graph $\mathcal{G}_t$ while maintaining the classification performance on the previous graphs $\{\mathcal{G}_1, \ldots, \mathcal{G}_{t-1}\}$.

5.3.2 Class & Task Incremental Settings of GCL

Depending on whether the task indicator is provided at the testing stage, GCL can be further divided into two settings: Class-Incremental Learning (CIL) and Task-Incremental Learning (TIL). Assume that each task in $\{\mathcal{G}_1, \ldots, \mathcal{G}_T\}$ has the same number of C classes and we use C_t to represent the unique set of classes in $\mathcal{G}_t$. In CIL, after learning all the T tasks, the model is required to classify each test instance into one of all the learned $T \times C$ classes. While under the TIL setting, the task indicator t is also provided with the test instance. Thus, the model is only required to assign a class to the test instance within C_t of task t. Compared to TIL, CIL is more practical yet more challenging as the label prediction space contains all the learned classes so far. In this chapter, we evaluate the proposed method under both settings to demonstrate its effectiveness.

5.3.3 Memory Replay in Graph Continual Learning

In GCL, a GNN model is sequentially trained across the task sequence $\{\mathcal{G}_1, \ldots, \mathcal{G}_T\}$. Typically, the model only has access to $\mathcal{G}_t = (A_t, X_t)$ at task t. To continually accommodate the new graph and alleviate the catastrophic forgetting, memory replay-based methods store representative data of previous $t-1$ tasks into a memory buffer $\mathcal{B}_t$. Then, the memory data are replayed with the data of task t to fit the new graph data and maintain the knowledge of previous tasks simultaneously. Therefore, the training objective of memory replay at task t can be formulated as follows:

$$\mathcal{L} = \underbrace{\ell(f_\theta(\mathcal{G}_t), Y_t)}_{\text{current task loss}} + \lambda \underbrace{\ell(f_\theta(\mathcal{B}_t), Y_{\mathcal{B}_t})}_{\text{memory loss}}, \tag{5.1}$$

where $f_\theta(\cdot)$ is the GNN model parameterized by θ, $Y_{\mathcal{B}_t}$ denote the labels of nodes in the memory buffer $\mathcal{B}_t$ (*i.e.*, all class labels encountered in the previous $t-1$ tasks), $\ell(\cdot)$ denotes a loss function, *i.e.*, cross-entropy loss, and λ is a hyperparameter to control the importance of the memory loss.

The memory buffer plays an important role in maintaining previously learned knowledge and different memory construction methods have been proposed. For example, ERGNN [54] sampled the representative nodes of the previous tasks as the memory. However, the rich topological information is neglected in ERGNN. Other approaches like SSM [63] aim to utilize the structure information by sparsifying the neighbors of the sampled nodes based on their topological importance. Then, the important neighborhood structures together with the sampled nodes are used to construct the memory. Despite the success achieved by these methods, the memory buffer constructed with partial graph data fails to preserve the holistic semantics of the original graph and it may lead to privacy leakage issues when the sampled nodes/edges are sensitive data. Further, the memory budget is typically kept significantly smaller than the current graph data, so it can lead to class imbalance between the memory data and the current graph data. Our method DeLoMe is proposed to

tackle these three issues.

5.4 Debiased Lossless Memory Replay

5.4.1 Lossless Memory Learning

Key intuition

Instead of using partial graph data as memory replay data, we propose to learn a small set of prototypical node representatives as the memory data which can holistically represent the original graph structure and attributes. Taking the task $t-1$ as an example, given $\mathcal{G}_{t-1} = (A_{t-1}, X_{t-1})$ with the label set Y_{t-1}, we aim to learn a *compressed graph* $\hat{\mathcal{G}}_{t-1} = (I, \hat{X}_{t-1})$ associated with label set $\hat{Y}_{t-1}$ as our memory data, where the fixed identity matrix I represents the structure of $\hat{\mathcal{G}}_{t-1}$ with learned prototypical nodes. Note that the size of $\hat{X}_{t-1}$ is constrained by the memory budget and is significantly smaller than X_{t-1}.

Since the learned $\hat{\mathcal{G}}_{t-1}$ captures the holistic semantics of the original graph $\mathcal{G}_{t-1}$ at task $t-1$, we can expect that a GNN model trained on $\hat{\mathcal{G}}_{t-1}$ would achieve comparable performance to that trained on $\mathcal{G}_{t-1}$. Following this idea, the learning objective of $\hat{\mathcal{G}}_{t-1}$ can be formulated as follows:

$$\min_{\hat{\mathcal{G}}_{t-1}} \ell(f_{\hat{\theta}}(\mathcal{G}_{t-1}), Y_{t-1}), \quad \text{s.t.} \quad \hat{\theta} = \arg\min_{\theta} \ell(f_{\theta}(\hat{\mathcal{G}}_{t-1}), \hat{Y}_{t-1}), \tag{5.2}$$

where $\hat{\theta}$ denotes the parameters of the graph neural network $f(\cdot)$ trained on $\hat{\mathcal{G}}_{t-1}$. Due to the nested loop optimization of Eq. (5.2), directly solving the above objective would be prohibitively expensive. To address this challenge, one-step gradient matching [128] is proposed, which aims to match the gradient of the same model with regard to the real data and the learned data at the first training epoch. Inspired by this, the optimization objective Eq. (5.2) can be transformed into a lossless compression as follows:

$$\min_{\hat{\mathcal{G}}_{t-1}} d(\nabla_{\theta} \ell(f_{\theta}(\mathcal{G}_{t-1}), Y_{t-1}), \nabla_{\theta} \ell(f_{\theta}(\hat{\mathcal{G}}_{t-1}), \hat{Y}_{t-1})), \tag{5.3}$$

where $d(\cdot)$ is a distance function to measure the difference between the two gradients. Moreover, to get more generalized $\hat{\mathcal{G}}_{t-1}$ without fitting to a specific model initialization, we aim to devise an objective of Eq. (5.3) that can be minimized under different random initializations of the $f_\theta(\cdot)$. Thus, the final objective to learn $\hat{\mathcal{G}}_{t-1}$ is defined as follows:

$$\min_{\hat{\mathcal{G}}_{t-1}} \sum_{\theta_p \sim \Theta} d(\nabla_{\theta_p} \ell(f_{\theta_p}(\mathcal{G}_{t-1}), Y_{t-1}), \nabla_{\theta_p} \ell(f_{\theta_p}(\hat{\mathcal{G}}_{t-1}), \hat{Y}_{t-1}))\,, \tag{5.4}$$

where θ_p is a random instantiation of the parameter space Θ.

By optimizing Eq. (5.4), we obtain the compressed graph $\hat{\mathcal{G}}_{t-1}$, in which a small set of prototypical nodes are learned for each class in $\mathcal{G}_{t-1}$. To achieve the lossless compression, these prototypical nodes are enforced to capture diverse important class-level structure and attribute information. $\hat{\mathcal{G}}_{t-1}$ is then used to update the memory buffer $\mathcal{B}_{t-1}$, *i.e.*, $\mathcal{B}_t = \mathcal{B}_{t-1} \cup \hat{\mathcal{G}}_{t-1}$. Then, $\mathcal{B}_t$ is replayed with the graph $\mathcal{G}_t$ when learning the next task t (see Figure 5.2).

The learning algorithm

To obtain the memory $\hat{\mathcal{G}}_{t-1} = (I, \hat{X}_{t-1})$ with $\hat{Y}_{t-1}$ of task $t-1$, we need to learn the node feature $\hat{X}_{t-1}$ and label set $\hat{Y}_{t-1}$. Since $\hat{Y}_{t-1}$ represents the node label and is discrete, $\hat{Y}_{t-1}$ is fixed as the same classes as the original label set Y_{t-1}. Therefore, we only need to learn $\hat{X}_{t-1}$ for task $t-1$. To accelerate the learning process, let b be the memory budget for each class, $\hat{X}_{t-1}$ is initialized as the features of randomly selected b nodes from each class in $\mathcal{G}_{t-1}$. To further reduce the computation cost, we sample a fixed number of neighbors for each node in $\mathcal{G}_{t-1}$ at each hop, and adopt the mini-batch training strategy. In addition, the gradient matching and learning of $\hat{X}$ is performed on each class separately. Specifically, for a given class c in Y_{t-1}, a batch of nodes belonging to class c is randomly sampled from $\mathcal{G}_{t-1}$ together with the associated neighborhoods, which is denoted as $\mathcal{G}^c_{t-1} = (A^c_{t-1}, X^c_{t-1})$. Meanwhile, we

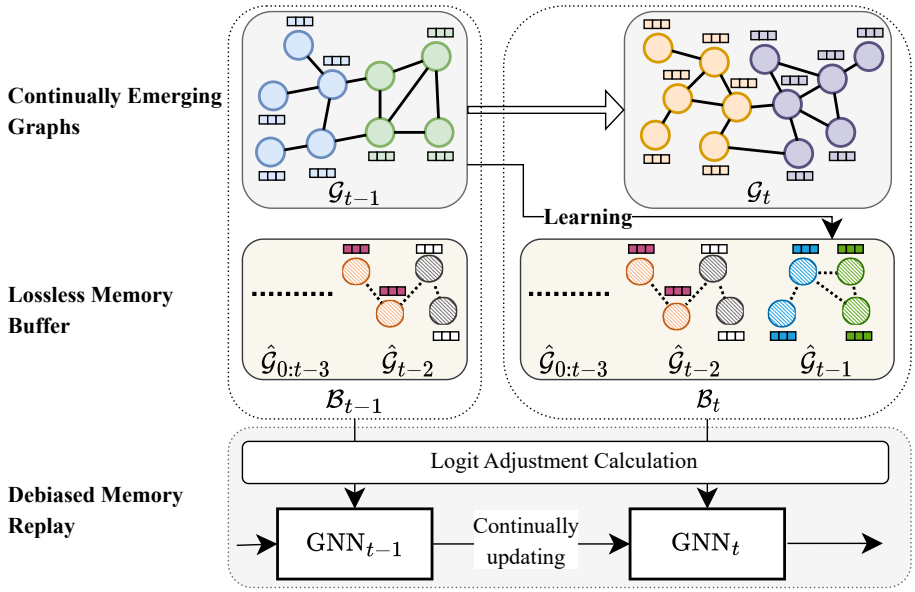


Figure 5.2 : Overview of DeLoMe. We take two consecutive tasks ($\mathcal{G}_{t-1}$ and $\mathcal{G}_t$) as an example, where GNN_{t-1} and GNN_t represent the GNN model trained after task $t-1$ and t respectively. At task $t-1$, we learn prototypical node representation-based memory $\hat{\mathcal{G}}_{t-1}$ for $\mathcal{G}_{t-1}$ and add it to the memory buffer via $\mathcal{B}_t = \mathcal{B}_{t-1} \cup \hat{\mathcal{G}}_{t-1}$. At task t, the memory buffer $\mathcal{B}_t$ is replayed with the current graph data $\mathcal{G}_t$ to train the model GNN_t using our debiased GCL objective. The process is repeated until all the tasks are learned.

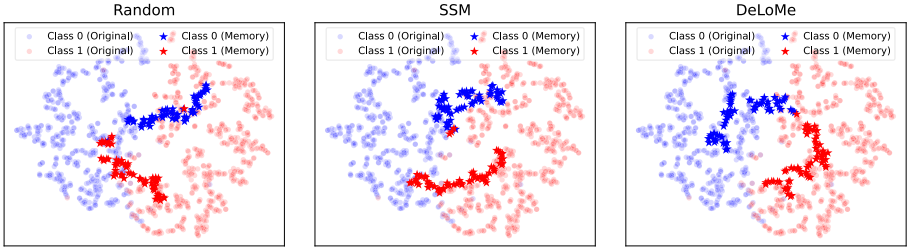


Figure 5.3 : Visualization of node embeddings of the original graph and the memories obtained by different methods on this graph.

get the corresponding nodes of class c from $\hat{\mathcal{G}}_{t-1}$ and denote it as $\hat{\mathcal{G}}^c_{t-1} = (I, \hat{X}^c_{t-1})$. Then, the sampled $\mathcal{G}^c_{t-1}$ and $\hat{\mathcal{G}}^c_{t-1}$ are fed into the same GNN model to calculate the gradient matching loss. Finally, $\hat{X}^c_{t-1}$ is optimized via gradient descent to minimize the graph matching loss. Note that the graph neural network $f_\theta(\cdot)$ is not updated during the learning process and we adopt different initializations of $f_\theta(\cdot)$ to learn a generalized $\hat{\mathcal{G}}_{t-1}$. By imitating the training trajectory of the original graph $\mathcal{G}_{t-1}$, $\hat{\mathcal{G}}_{t-1} = (I, \hat{X}_{t-1})$ is enforced to capture the holistic semantics of $\mathcal{G}_{t-1}$ and the global structure information is implicitly incorporated into $\hat{X}_{t-1}$. The algorithm of lossless memory learning at the task $t-1$ in Algorithm 3.

As illustrated in Figure 5.3 where we visualize the node embeddings of the memories constructed by two sampling methods (random node sampling and SSM) and DeLoMe, the prototypical node representations in the memory learned by DeLoMe can better preserve the distribution of the nodes of different classes in the original graph, compared to the other two methods. This indicates the better capability of DeLoMe in capturing more holistic graph semantics.

Time complexity analysis

We analyze the time complexity of obtaining $\hat{\mathcal{G}}_{t-1} = (I, \hat{X}_{t-1})$ at task $t-1$. As mentioned above, the learning process is conducted for each class $\mathcal{G}^c_{t-1} = (A^c_{t-1}, X^c_{t-1})$ separately. We take a two-layer SGC [112] as the GNN model for gradient matching

Algorithm 3 Memory Learning for graph data of task $t-1$

Input: Graph data $\mathcal{G}_{t-1} = (A_{t-1}, X_{t-1})$ with label Y_{t-1}, memory budget b, graph neural network $f_\theta(\cdot)$, learning rate η, and the number of epochs E.
Output: Memory data $\hat{\mathcal{G}}_{t-1} = (I, \hat{X}_{t-1})$ with label $\hat{Y}_{t-1}$
Set $\hat{Y}_{t-1}$ to fixed class values as in Y_{t-1}, initialize $\hat{X}$ by randomly selecting b nodes from each class in $\mathcal{G}_{t-1}$.
for $e = 1, \ldots, E$ **do**
    Initialize graph neural network parameter θ_p from Θ
    for $c = 1, \ldots, C$ **do**
        Sample $\mathcal{G}^c_{t-1} = (A^c_{t-1}, X^c_{t-1})$ and Y^c_{t-1} from $\mathcal{G}_{t-1}$
        Sample $\hat{\mathcal{G}}^c_{t-1} = (I, \hat{X}^c_{t-1})$ and $\hat{Y}^c_{t-1}$ from $\hat{\mathcal{G}}_{t-1}$
        Compute gradient: $G^c_{t-1} = \nabla_{\theta_p} \ell(f_{\theta_p}(\mathcal{G}^c_{t-1}), Y^c_{t-1})$
        Compute gradient: $\hat{G}^c_{t-1} = \nabla_{\theta_p} \ell(f_{\theta_p}(\hat{\mathcal{G}}^c_{t-1}), \hat{Y}^c_{t-1})$
        Update $\hat{X} = \hat{X} - \eta \nabla_{\hat{X}} D(G^c_{t-1}, \hat{G}^c_{t-1})$
    end for
end for

and denote the number of nodes and edges in $\mathcal{G}^c_{t-1}$ as N^n_c and N^e_c respectively. The dimension of the node features is denoted as F. For the two-layer SGC, the time complexity of forward and backward propagation with regard to $\mathcal{G}^c_{t-1}$ and $\hat{\mathcal{G}}^c_{t-1}$ are $\mathcal{O}(4N^e_c F + 2N^n_c FC))$ and $\mathcal{O}(2bFC)$ respectively. Given the training epochs E, the time complexity for class c is $\mathcal{O}(4N^e_c EF + 2N^n_c EFC + 2bEFC)$. Then, the overall time complexity of learning $\hat{\mathcal{G}}_{t-1}$ is $\mathcal{O}(\sum_{c=1}^{C}(4N^e_c EF + 2N^n_c EFC + 2bEFC))$. Since we adopt the minibatch training strategy and sample a fixed number of neighbors for each node at each hop, the numbers of N^n_c and N^e_c are typically small and do not induce high computation. Besides, the learning process can be implemented in parallel in practice to further reduce the learning time.

5.4.2 Debiased Memory Replay

Although the learned memory data are lossless compared to previous graphs, there is a data imbalance between the classes in the new graph $\mathcal{G}_t$ and that in the memory buffer $\mathcal{B}_t$. This is because the memory budget b of each class is typically much smaller than the number of nodes belonging to new classes in $\mathcal{G}_t$. This class imbalance would increase, when the graph evolves with more current graph data or a tighter memory budget is given, amplifying the degraded performance for GCL. To tackle this problem, we propose a debiased memory replay method that adjusts the prediction logits of the classes in the memory data and the current graph data based on the class label frequencies during the memory replay. Specifically, at task t, we have the memory buffer $\mathcal{B}_t = \{\hat{\mathcal{G}}_1, \ldots, \hat{\mathcal{G}}_{t-1}\}$, and the vanilla objective of memory replay in Eq. (5.1) can be explicitly formulated as:

$$\mathcal{L} = \ell(H_t, Y_t) + \lambda \sum_{j=1}^{t-1} \ell(\hat{H}_j, \hat{Y}_j)\,, \tag{5.5}$$

where H_t and $\hat{H}_j$ are the prediction logits of $f_\theta(\cdot)$ for the nodes in $\mathcal{G}_t$ and $\hat{\mathcal{G}}_j$ respectively. Directly minimizing Eq. (5.5) would make the model biased toward the current task as the current graph data $\mathcal{G}_t$ dominates the training data. We address this problem by calibrating the logits H_t and $\hat{H}_j$ based on their label frequencies. Given the memory budget b, the calibration magnitude for each class in the memory buffer is equal and can be defined as:

$$\Pi_{\mathcal{B}_t} = \tau \log \frac{b}{|Y_t| + (t-1)bC}\,, \tag{5.6}$$

where we assume each task contains the same C classes, τ is a scaling hyperparameter, and $|Y_t|$ returns the number of samples in Y_t. For a class c in the current graph $\mathcal{G}_t$, the calibration magnitude is defined as:

$$\Pi_t^c = \tau \log \frac{|Y_t^c|}{|Y_t| + (t-1)bC}\,, \tag{5.7}$$

where $|Y_t^c|$ denotes the number of training samples of class c in $\mathcal{G}_t$. By incorporating these two calibrations into Eq. (5.5), our debiased GCL training loss is as follows:

$$\mathcal{L} = \sum_{c=1}^{C} \ell(H_t^c + \Pi_t^c, Y_t^c) + \sum_{j=1}^{i-1} \ell(\hat{H}_j + \Pi_{\mathcal{B}}, \hat{Y}_j)\,, \tag{5.8}$$

where we discard the weight parameter λ to simplify the parameter selection. Compared to Eq. (5.5), Eq. (5.8) augments the softmax cross-entropy with a pairwise margin based on label frequencies [133, 134]. In this way, the predictions for dominant classes in the current graph do not overwhelm those for tail classes in the memory data, thus reducing the bias toward the dominant classes in $\mathcal{G}_t$. The detailed training steps of DeLoMe are presented in Algorithm 4.

Algorithm 4 DeLoMe

Input: A sequence of graph learning tasks: $\{\mathcal{G}_1, \ldots, \mathcal{G}_T\}$ and an empty memory buffer $\mathcal{B}_1$ with a budget b for each class.
Output: A continually learned graph neural network $f_\theta(\cdot)$
Initialize $f_\theta(\cdot)$.
for $t = 1, \ldots, T$ **do**
  if $t > 1$ **then**
    Update memory buffer $\mathcal{B}_{t-1}$ with $\hat{\mathcal{G}}_{t-1}$ (Eq.(4)), *i.e.*, $\mathcal{B}_t = \mathcal{B}_{t-1} \cup \hat{\mathcal{G}}_{t-1}$
    Calculate the logit adjustments via Eq.(6) and Eq.(7)
  end if
  Update $f_\theta(\cdot)$ by minimizing training objective Eq.(8)
  Obtain $\hat{\mathcal{G}}_t$ with Algorithm 3
end for

5.5 Experiments

5.5.1 Datasets

Following the GCL benchmark [10], four public graph datasets are employed, *i.e.*, CoraFull [135], Arxiv [136], Reddit [5] and Products [136]. Specifically, CoraFull and Arxiv are citation networks, Reddit is constructed from Reddit posts, and Products is a co-purchasing network from Amazon. For all datasets, each task is set to contain only two classes [10]. Besides, for each class, the proportions of training, validation and testing are set to be 0.6, 0.2 and 0.2, respectively. The details on these datasets are listed as follows:

- **CoraFull**[*]: It is a citation network containing 70 classes, where nodes represent papers and edges represent citation links between papers.
- **Arxiv**[†]: It is also a citation network between all Computer Science (CS) ARXIV papers indexed by MAG [137]. Each node in Arxiv denotes a CS paper and the edge between nodes represents a citation between them. The nodes are classified into 40 subject areas. The node features are computed as the average word-embedding of all words in the title and abstract.
- **Reddit**[‡]: It encompasses Reddit posts generated in September 2014, with each post classified into distinct communities or subreddits. Specifically, nodes represent individual posts, and the edges between posts exist if a user has commented on both posts. Node features are derived from various attributes, including post title, content, comments, post score, and the number of comments.

[*] `https://docs.dgl.ai/en/1.1.x/generated/dgl.data.CoraFullDataset.html`

[†] `https://ogb.stanford.edu/docs/nodeprop/\#ogbn-arxiv`

[‡] `https://docs.dgl.ai/en/1.1.x/generated/dgl.data.RedditDataset.html\#dgl.data.RedditDataset`

- **Products**[§]: It is an Amazon product co-purchasing network, where nodes represent products sold in Amazon and the edges between nodes indicate that the products are purchased together. The node features are constructed with the dimensionality-reduced bag-of-words of the product descriptions.

The statistics of these datasets are summarized in the Table 5.1.

Table 5.1 : Statistics of the graph datasets.

Datasets	CoraFull	Arxiv	Reddit	Products
# nodes	19,793	169,343	227,853	2,449,028
# edges	130,622	1,166,243	114,615,892	61,859,036
# classes	70	40	40	46
# tasks	35	20	20	23
# Avg. nodes per task	660	8,467	11,393	122,451
# Avg. edges per task	4,354	58,312	5,730,794	2,689,523

5.5.2 Baseline Models

Two categories of state-of-the-art (SOTA) continual learning methods are employed for comparison. The first category contains traditional continual learning methods, *i.e.*, EWC [125], LwF [138], GEM [139] and MAS [140]. The second category includes four SOTA GCL methods: ERGNN [54], TWP [55], HPNs [141], SSM [61], SEM [63] and CaT [64]. In addition, we include two other methods: Joint and Fine-tune. The Joint method is an oracle model that can see all graphs at all times and performs GCL on the full graphs of all tasks, while Fine-tune is a baseline that simply fine-tunes the learned model from previous tasks without continual learning techniques.

[§]`https://ogb.stanford.edu/docs/nodeprop/\#ogbn-products`

5.5.3 Evaluation Metrics

Average accuracy (AA) and average forgetting (AF) are adopted to evaluate the model's performance. Specifically, AA and AF are calculated from the accuracy matrix $M \in \mathbb{R}^{T\times T}$, where T is the number of tasks. The entry $M_{tj}(t \geqslant j)$ denotes the classification accuracy on task j after the model is optimized on task t. After learning all the T tasks, the overall AA and AF can be calculated as follows:

$$\mathrm{AA} = \frac{\sum_{j=1}^{T} M_{Tj}}{T}, \quad \mathrm{AF} = \frac{\sum_{j=1}^{T-1}(M_{Tj} - M_{jj})}{T-1}. \tag{5.9}$$

To sum up, AA evaluates the average performance of the model on all the learned tasks after learning the current task, and AF describes how the performance of previous tasks is affected by the current task. For both AA and AF, the higher value denotes the better GCL performance.

5.5.4 Implementation Details

To have a fair comparison, all the continual learning methods, including the proposed method, are implemented based on the graph continual learning benchmark [10]. For DeLoMe, the memory budget is also set as the same in [63], *i.e.*, 60 per class for the CoraFull dataset and 400 per class for the other datasets. For ERGNN [54], the memory budget is set to be up to 800 per class to demonstrate the advantage of the proposed method. For SSM [61] and [63], we conduct experiments with the same memory budgets. However, to preserve the topological information, SSM and SEM need to store the neighbors of selected nodes. The number of neighbors for each node to store is set to 5 at each hop for both SSM and SEM.

Different GNN backbones, such as GCN [98] and SGC [112], can be applied to these continual learning methods. To have a fair comparison to the baselines [63], we employ a two-layer SGC model as the backbone. Specifically, the hidden dimension is set to 256 for all methods. The number of training epochs of each graph learning

task is 200 with Adam as the optimizer and the learning rate is set to 0.005.

For the lossless memory learning in the proposed method, we employ the same two-layer SGC model as the GNN model to calculate the gradient-matching loss. The prototypical node representations are set as learnable parameters and are initialized with the original node attributes. The labels of the prototypical nodes are fixed during the learning process. Adam is used as the optimizer to learn the prototypical representations. The learning epochs and learning rate are set to 800 and 0.0001 for all tasks and datasets, respectively. The scaling parameter τ in the debiased loss function is set to 1 for all experiments for simplicity.

The code is implemented with Pytorch (version: 1.10.0), DGL (version: 0.9.1), OGB (version: 1.3.6), and Python 3.8.5. Moreover, all experiments are conducted on a Linux server with an Intel CPU (Intel Xeon E-2288G 3.7GHz) and an Nvidia GPU (Quadro RTX 6000). For each dataset, we report the average performance with standard deviations after 5 runs with different seeds under both task incremental and class incremental settings.

5.5.5 Main Results

Results under class-incremental learning (CIL)

The results of all methods under the CIL setting are shown in Table 5.2. Note that the results of baselines are taken from the paper [63] since we adopt the same GCL benchmark [10][¶]. From the table, we can draw the following observations: (1) Directly fine-tuning the learned model from previous tasks on the current task data leads to serious performance degradation because the knowledge of previous tasks could be easily overwritten by the new tasks. (2) Continual learning methods proposed for Euclidean data generally do not achieve satisfactory performance for GCL,

[¶] `https://github.com/QueuQ/CGLB/tree/master`

Table 5.2 : Results (mean±std) under the class-incremental learning setting on four datasets. Fine-tune and Joint are shown to respectively serve as approximated lower bound and upper bound performance. The best performance achieved by continual learning methods on each dataset is highlighted in bold. "↑" denotes the higher value represents better performance.

Methods	CoraFull		Arxiv		Reddit		Products	
	AA/%↑	AF/%↑	AA/%↑	AF/%↑	AA/%↑	AF/%↑	AA/%↑	AF/%↑
Fine-tune	3.5±0.5	-95.2±0.5	4.9±0.0	-89.7±0.4	5.9±1.2	-97.9±3.3	7.6±0.7	-88.7±0.8
Joint	81.2±0.4	-	51.3±0.5	-	97.1±0.1	-	71.5±0.1	-
EWC	52.6±8.2	-38.5±12.1	8.5±1.0	-69.5±8.0	10.3±11.6	-33.2±26.1	23.8±3.8	-21.7±7.5
MAS	6.5±1.5	-92.3±1.5	4.8±0.4	-72.2±4.1	9.2±14.5	-23.1±28.2	16.7±4.8	-57.0±31.9
GEM	8.4±1.1	-88.4±1.4	4.9±0.0	-89.8±0.3	11.5±5.5	-92.4±5.9	4.5±1.3	-94.7±0.4
LwF	33.4±1.6	-59.6±2.2	9.9±12.1	-43.6±11.9	86.6±1.1	-9.2±1.1	48.2±1.6	-18.6±1.6
TWP	62.6±2.2	-30.6±4.3	6.7±1.5	-50.6±13.2	8.0±5.2	-18.8±9.0	14.1±4.0	-11.4±2.0
ERGNN	34.5±4.4	-61.6±4.3	21.5±5.4	-70.0±5.5	82.7±0.4	-17.3±0.4	48.3±1.2	-45.7±1.3
SSM-uniform	73.0±0.3	-14.8±0.5	47.1±0.5	-11.7±1.5	94.3±0.1	-1.4±0.1	62.0±1.6	-9.9±1.3
SSM-degree	75.4±0.1	-9.7±0.0	48.3±0.5	-10.7±0.3	94.4±0.0	-1.3±0.0	63.3±0.1	-9.6±0.3
SEM-curvature	77.7±0.8	-10.0±1.2	49.9±0.6	-8.4±1.3	96.3±0.1	-0.6±0.1	65.1±1.0	-9.5±0.8
CaT	80.4±0.5	-5.3±0.4	48.2±0.4	-12.6±0.7	97.3±0.1	-0.4±0.0	**70.3±0.9**	**-4.5±0.8**
DeLoMe (Ours)	**81.0±0.2**	**-3.3±0.3**	**50.6±0.3**	**5.1±0.4**	**97.4±0.1**	**-0.1±0.1**	67.5±0.7	-17.3±0.3

Table 5.3 : Full results (mean±std) under the task-incremental learning setting.

Methods	CoraFull		Arxiv		Reddit		Products	
	AA/%↑	AF/%↑	AA/%↑	AF/%↑	AA/%↑	AF/%↑	AA/%↑	AF/%↑
Fine-Tune	56.0±4.2	-41.0±4.5	56.2±2.6	-36.2±2.6	79.5±24.2	-11.7±4.8	64.4±3.8	-31.1±4.4
Joint	95.5±0.2	-	90.3±0.4	-	99.5±0.0	-	95.3±0.8	-
EWC	89.8±1.0	-5.1±0.5	71.5±0.6	-0.9±0.6	83.9±15.1	-2.0±1.5	87.0±1.4	-1.7±1.2
MAS	92.2±0.9	-3.7±1.3	72.7±2.6	-18.5±2.5	61.1±7.1	-0.5±1.0	80.6±4.3	-13.7±3.7
GEM	91.5±0.5	-1.9±0.9	81.1±1.7	-4.0±1.8	98.9±0.1	-0.5±0.1	87.7±1.8	-7.0±2.0
LwF	93.8±0.1	-0.4±0.1	71.1±3.2	-1.5±0.8	98.6±0.1	-0.0±0.0	86.3±0.2	-0.5±0.1
TWP	94.3±0.9	-1.6±0.4	89.4±0.4	0.0±0.3	78.0±18.5	-0.2±0.4	81.8±3.3	-0.3±0.8
HPNs	-	-	85.8±0.7	**0.6±0.9**	-	-	80.1±0.8	2.9±1.0
ERGNN	86.3±1.0	-9.2±0.9	86.4±0.3	0.5±0.6	97.4±0.2	**4.7±0.1**	86.4±0.0	**11.7±0.0**
SSM-uniform	95.3±0.5	0.2±0.5	88.5±0.6	-1.3±0.5	99.2±0.0	-0.2±0.0	93.1±0.8	-1.8±0.3
SSM-degree	95.8±0.3	0.6±0.2	88.4±0.3	-1.1±0.1	99.3±0.0	-0.2±0.0	93.2±0.7	-1.9±0.0
SEM-curvature	**95.9±0.5**	0.7±0.4	89.9±0.3	-0.1±0.5	99.3±0.0	-0.2±0.0	93.2±0.7	-1.8±0.4
CaT	95.0±0.2	1.6±0.7	90.3±0.3	0.3±0.4	99.2±0.0	0.0±0.0	94.7±0.1	-0.0±0.1
DeLoMe (Ours)	95.4±0.1	**2.0±0.6**	**90.4±0.3**	-1.1±0.2	**99.4±0.0**	-0.1±0.0	**94.8±0.1**	-2.2±0.2

which verifies the fact that the unique graph properties should be taken into consideration for GCL. (3) Replay-based graph continual learning methods achieve much better performance than other baselines. Among them, SSM and SEM outperform ERGNN on all datasets. The reason could be attributed to that SSM and SEM preserve the topological information for the historical graph data in memory while ERGNN only stores the individual nodes. (4) Differently, CaT and DeLoMe propose to learn the memory and capture the semantics of the original graph. The performance gain of CaT and DeLoMe over SSM and SEM demonstrates that the learned memory buffer is more informative (*e.g.*, in capturing holistic graph information) and enhances the power of replaying memory. (5) Our method DeLoMe achieves new SOTA performance in nearly all cases. Its performance can even match or outperform the ideal method Joint on the CoraFull and Reddit datasets. Note that on Products, DeLoMe outperforms all methods except CaT. This may be attributed to that the node class frequency information in Products does not help accurately capture the imbalance bias due to the possible presence of less informative nodes in the largest dataset, rendering our debiased component less effective.

Results under task-incremental learning (TIL)

In Table 5.3, we report the results under the TIL setting, in which an SOTA TIL method, HPNs, is also included for comparison. From the table, we can first see that all the methods achieve much better performance under the TIL setting. This is because the availability of task indicators during inference makes TIL much easier than CIL. Despite the performance of our DeLoMe being slightly inferior to SEM [63] on the CoraFull dataset, DeLoMe achieves comparable performance with the Oracle model, Joint. For the other three datasets, DeLoMe achieves the best performance and even outperforms Joint on Arxiv dataset, which verifies the advantages of our two components in overcoming the forgetting and class imbalance problems.

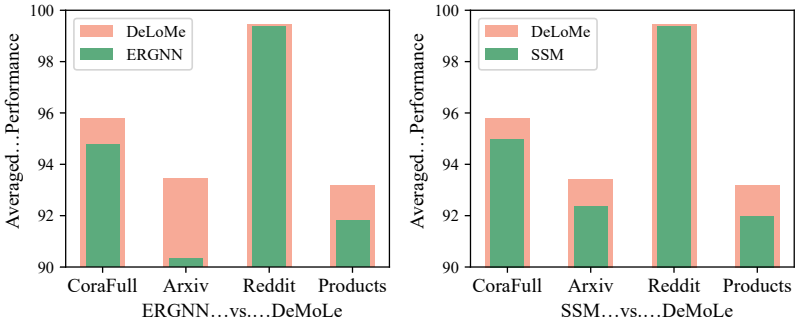


Figure 5.4 : Average accuracy (AA) of DeLoMe against SOTA sampling-based memory construction methods on all tasks.

5.5.6 Cost-effective Learning of Expressive Memory

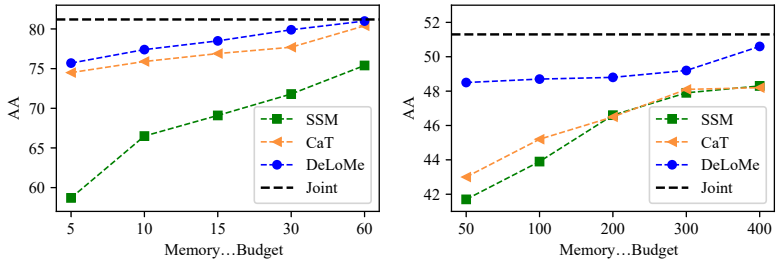
Expressiveness of sampling- vs. learning-based memory

In this subsection, we evaluate the expressiveness of the memory constructed by our learning-based method against two SOTA sampling-based methods, ERGNN and SSM. To evaluate the memory expressiveness, for each task, we train a GNN model with the memory data and calculate the node classification accuracy on the test set of the original graph. We follow the same experimental setting in the above GCL experiments and report the average classification accuracy of all tasks in Figure 5.4. It is clear that our learning-based method achieves much better performance than both sampling-based methods, achieving improvement by up to 3% in AA. This demonstrates the superiority of capturing the semantics of the original graph when constructing memory and explains the performance gain over the sampling-based methods in the GCL experiments.

Memory budget efficiency

We evaluate the performance of the proposed method with different memory budgets, with two best-baseline methods – SSM and CaT – and the oracle model Joint as the baselines. Due to the page limits, we report the results on CoraFull

and Arxiv under CIL. The memory budgets vary in $\{5, 10, 15, 30, 60\}$ for CoraFull and $\{50, 100, 200, 300, 400\}$ for Arxiv. The results are shown in Figure 5.5, which demonstrates that the learning-based methods (DeLoMe and CaT) are more effective than the sampling-based method with tight memory budgets. Compared to SSM and CaT, DeLoMe performs much more stably and achieves consistently better performance with varying budgets, especially on the Arxiv dataset. These results reinforce our empirical evidence on the effectiveness of DeLoMe in constructing memory and handling the class imbalance problem.



(a) CoraFull (b) Arxiv

Figure 5.5 : AA results with different memory budgets on CoraFull and Arxiv under the class-incremental learning.

Computational efficiency

We further investigate the memory construction and inference times of DeLoMe and employ SSM and CaT for comparison. Specifically, we report the average memory construction time per task and the overall inference time of each method on the largest dataset, Products. From the results in Table 5.4, we can see that CaT and DeLoMe require almost the same time for memory construction, while SSM is much faster than the two learning-based methods. This is because CaT and DeLoMe involve model optimization to enhance the memory to capture the holistic semantics of the original graph, while SSM employs the parameter-free sampling

strategy. This also explains the superiority of CaT and DeLoMe over SSM in AA and AF in Tables 5.2 and 5.3. In terms of the inference time, the three methods are nearly the same since the memory construction only happens in the training stage and the test setting remains the same for all methods.

Table 5.4 : Time (second) of different stages on Products.

Stage	SSM	CaT	DeloMe
Memory Construction	0.53	28.94	28.37
Inference	1.73	1.75	1.71

5.5.7 Ablation Study

We evaluate the contribution of two key components in DeLoMe, *i.e.*, lossless memory learning and debiased GCL learning. Specifically, we derive four variants. When the lossless memory learning is not exploited, we employ the sampling strategy to construct the memory as in ERGNN. Without loss of generality, we conduct the experiments on the CoraFull and Arxiv datasets under the CIL setting. The results of different variants are shown in Table 5.5. From the table, we can see that adding either of the two components contributes to a significant improvement compared to the variant that does not use both components. These showcase the importance of both components, as well as their effectiveness in addressing the memory expressiveness and data imbalance problems. In general, having a more expressive memory helps achieve larger improvements than tackling the data imbalance problem. Nevertheless, with our biased GCL objective, DeLoMe can achieve further large improvement over using our expressive memory learning component alone.

5.5.8 Results with GCN as Backbone

As stated above, different GNNs can be used as the backbone of DeLoMe. In the above, we report the results with SGC [112] as the backbone. In this subsection,

Table 5.5 : Ablation study of the two components in DeLoMe.

Lossless Memory	Debiased Learning	CoraFull		Arixv	
		AA↑	AF↑	AA↑	AF↑
×	×	36.9	-59.0	24.6	-66.1
×	✓	50.2	-40.6	33.9	-31.1
✓	×	78.5	-9.3	47.9	-15.8
✓	✓	81.0	-3.3	50.6	5.1

we further present the results with GCN [98] as the backbone. Specifically, we employ the same baselines for comparison, and the results under the class- and task-incremental learning are shown in Table 5.6 and Table 5.7 respectively. From the results, we can draw similar observations as above, *i.e.*, the proposed DeLoMe can effectively overcome the catastrophic forgetting problem in graph continual learning by utilizing lossless memory and debiased memory replay. The results also verify the effectiveness of DeLoMe with different backbones.

5.6 Conclusion

In this chapter, we propose a novel memory replay-based GCL method, DeLoMe. Traditional replay-based graph continual learning methods typically construct the memory of the previous task using partial graph data, failing to preserve the holistic semantics of the original graph at each task. To tackle this issue, we learn compressed prototypical node representations as the memory by a gradient matching approach. In this way, the learned representations capture the holistic graph structure and attribute information. Besides, the learned representations help preserve the privacy of the graph data when replaying. To overcome the class imbalance problem between the learned memory and the new-coming graph, we further proposed a debiased memory replay objective by calibrating the prediction logits of the classes in both memory data and the current task based on the label frequencies. Extensive

Table 5.6 : Results under the class-incremental learning setting on four datasets with GCN as the backbone. Fine-tune and Joint are shown to respectively serve as approximated lower bound and upper bound performance. The best performance achieved by continual learning methods on each dataset is highlighted in bold. "↑" denotes the higher value represents better performance.

Methods	CoraFull		Arixv		Reddit		Products	
	AA/%↑	AF/%↑	AA/%↑	AF/%↑	AA/%↑	AF/%↑	AA/%↑	AF/%↑
Fine-tune	2.9±0.0	-94.7±0.1	4.9±0.0	-87.0±1.5	5.1±0.3	-94.5±2.5	3.4±0.8	-82.5±0.8
Joint	80.6±0.3	-	46.4±1.4	-	99.3±0.2	-	71.5±0.7	-
EWC	15.2±0.7	-81.1±1.0	4.9±0.0	-88.9±0.3	10.6±1.5	-92.9±1.6	3.3±1.2	-89.6±2.0
MAS	12.3±3.8	-83.7±4.1	4.9±0.0	-86.8±0.6	13.1±2.6	-35.2±3.5	15.0±2.1	-66.3±1.5
GEM	7.9±2.7	-84.8±2.7	4.8±0.5	-87.8±0.2	28.4±3.5	-71.9±4.2	5.5±0.7	-84.3±0.9
LwF	2.0±0.2	-95.0±0.2	4.9±0.0	-87.9±1.0	4.5±0.5	-82.1±1.0	3.1±0.8	-85.9±1.4
TWP	20.9±3.8	-73.3±4.1	4.9±0.0	-89.0±0.4	13.5±2.6	-89.7±2.7	3.0±0.7	-89.7±1.0
ERGNN	3.0±0.1	-93.8±0.5	30.3±1.5	-54.0±1.3	88.5±2.3	-10.8±2.4	24.5±1.9	-67.4±1.9
SSM-uniform	72.3±0.8	-15.5±1.5	45.1±1.1	-12.2±1.4	93.8±0.4	-2.0±0.3	61.8±1.2	-10.7±1.1
SSM-degree	74.4±0.2	-9.9±0.1	46.0±0.4	-11.3±0.8	94.0±0.2	-1.9±0.2	62.9±0.4	-10.3±0.5
SEM-curvature	79.6±0.3	-2.7±0.1	51.0±0.3	-6.7±0.6	95.1±0.6	-1.5±0.7	64.0±1.6	-11.2±1.8
CaT	79.5±0.4	-5.5±0.2	47.1±0.4	-13.7±0.2	98.0±0.1	-0.1±0.1	**70.6±1.0**	**-4.0±0.9**
DeLoMe (Ours)	**81.8±0.3**	**1.9±0.4**	**52.8±0.3**	**0.2±0.6**	**98.1±0.1**	**0.6±0.2**	67.0±0.8	-18.3±0.4

experiments on four datasets demonstrate the effectiveness of the proposed method under both class- and task-incremental learning settings of GCL.

Table 5.7 : Results under the task-incremental learning setting with GCN as the backbone.

Methods	CoraFull		Arixv		Reddit		Products	
	AA/%↑	AF/%↑	AA/%↑	AF/%↑	AA/%↑	AF/%↑	AA/%↑	AF/%↑
Fine-Tune	58.0±1.7	-38.4±1.8	61.7±3.8	-28.2±3.3	73.6±3.5	-26.9±3.5	67.6±1.6	-25.4±1.6
Joint	95.2±0.2	-	90.3±0.2	-	99.4±0.1	-	91.8±0.2	-
EWC	78.9±2.4	-15.5±2.3	78.8±2.7	-5.0±3.1	91.5±4.2	-8.1±4.6	90.1±0.3	-0.7±0.3
MAS	93.0±0.5	-0.6±0.2	88.4±0.2	-0.0±0.0	98.6±0.5	-0.1±0.1	91.2±0.6	-0.5±0.2
GEM	91.6±0.6	-1.8±0.6	87.3±0.6	2.8±0.3	91.6±5.6	-8.1±5.8	87.8±0.5	-2.9±0.5
LwF	56.1±2.0	-37.5±1.8	84.2±0.5	-3.7±0.6	80.9±4.3	-19.1±4.6	66.5±2.2	-26.3±2.3
TWP	92.2±0.5	-0.9±0.3	86.0±0.8	-2.8±0.8	87.4±3.8	-12.6±4.0	90.3±0.1	-0.5±0.1
ERGNN	90.6±0.1	-3.7±0.1	86.7±0.1	**11.4±0.9**	98.9±0.0	-0.1±0.1	89.0±0.4	-2.5±0.3
SSM-uniform	94.5±0.8	-0.1±0.9	88.8±1.2	-2.1±0.7	98.8±0.3	-0.9±0.6	90.9±2.8	-2.2±1.0
SSM-degree	94.3±1.1	0.9±0.5	88.1±1.3	-1.0±0.4	99.0±0.2	-0.4±0.2	91.1±0.9	-1.8±0.8
SEM-curvature	95.0±0.5	-0.2±0.8	88.8±0.1	-1.0±0.2	99.2±0.1	-0.1±0.3	91.6±0.5	-1.5±0.8
CaT	94.3±0.2	0.5±0.5	90.2±0.2	0.2±0.1	99.2±0.0	**0.1±0.1**	**94.3±0.6**	**0.1±0.2**
DeLoMe (Ours)	**95.2±0.3**	**1.3±0.1**	**90.6±0.3**	2.3±1.1	**99.3±0.1**	-0.2±0.1	94.2±0.1	-1.4±0.1

Chapter 6

Replay-and-Forget-Free Graph Class-Incremental Learning: A Task Profiling and Prompting Approach

6.1 Introduction

Graph continual learning (GCL) [10–12, 127] aims to continually learn a model that not only accommodates the new emerging graph data but also maintains the learned knowledge of previous graph tasks, with each graph task comprising nodes from a set of unique classes in a graph. Due to privacy concerns and the hardware limitations in storage and computation, GCL assumes that the data of previous graph tasks is not accessible when learning new graph tasks. This leads to *catastrophic forgetting* of the learned knowledge, *i.e.*, degraded classification accuracy on previous tasks due to model updating on new tasks.

Graph class-incremental learning (GCIL) is one key setting of GCL, in which task identifiers (IDs) are not provided during inference. *Graph task-incremental learning* (GTIL) is another GCL setting where the task ID is given for each test sample. As a result, a set of separate classifiers can be learned for different graph tasks in GTIL and the task-specific classifier can be used for each test sample. Compared to GTIL, the absence of task IDs in GCIL presents an additional challenge, known as *inter-task class separation* [14–16], *i.e.*, the class separation in one graph task is obstructed by the presence of classes from other tasks. Consequently, the classification performance in GCIL is typically far below that in GTIL [54, 55, 61, 63–65]. This chapter focuses on the GCIL setting – a more compelling GCL problem – aiming to bridge the

performance gap between GCIL and GTIL.

Existing GCL methods often alleviate the catastrophic forgetting through preserving important model parameters of previous tasks [55], continually expanding model parameters for new tasks [62, 141], or augmenting with a memory module for data replay [54, 61, 63–65, 65]. However, their ability to handle the inter-task class separation is limited, leading to poor GCIL classification accuracy, especially the accuracy of the previous graph tasks. Thus, existing GCL methods typically show substantially higher forgetting in GCIL than in GTIL.

To address these issues, we introduce a novel GCIL approach, namely **Task Profiling and Prompting (TPP)**. In particular, we reveal for the first time theoretically and empirically that the task ID of a test sample can be accurately predicted by using a Laplacian smoothing-based method to profile each graph task with a prototypical embedding. With the existence of edges between nodes, this task profiling method guarantees that the task prototypes of the same graph task are nearly the same with a large smoothing step, while those of different tasks are distinct due to differences in graph structure and node attributes. High task ID prediction accuracy helps confine the classification space of the test samples to the classes of the predicted task (*e.g.*, Task 1 or 2 in Figure 6.1b,c) instead of all the learned tasks (*e.g.*, both tasks as in Figure 6.1a), eliminating the inter-task class separation issue. There have been some studies on task ID prediction [14–16], but they are designed for Euclidean data that are i.i.d. (independent and identically distributed). As a result, they fail to leverage the graph structure and node attributes in the non-Euclidean graph data and are not suited for graph task ID prediction.

To address the catastrophic forgetting problem, we further propose a novel graph prompting approach for GCIL. Specifically, we optimize a single, small learnable prompt using a simple frozen pre-trained graph neural network (GNN) to capture

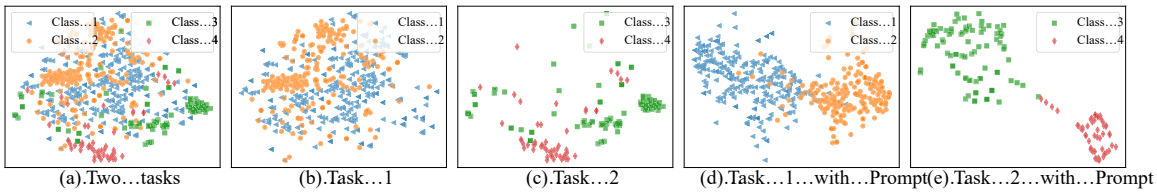


Figure 6.1 : **(a)** Classification space of two graph tasks when no task ID is provided. The classification space is split into two separate spaces in Task 1 in **(b)** and Task 2 in **(c)** when the task ID can be accurately predicted. This helps alleviate the inter-task class separation issue. To mitigate catastrophic forgetting, we learn a graph prompt for each task that absorbs task-specific discriminative information for better class separation within each task, as shown in **(d)** and **(e)** respectively. This essentially results in a separate classification model for each task, achieving fully forget-free GCIL models.

the task-specific knowledge for each graph task during training. Despite being small, the task-specific knowledge learned in the prompts can ensure the intra-task class separation, as shown in Figure 6.1d,e. At test time, given a test graph, the task prototype constructed with its structure and node attributes is utilized for task ID prediction, and the graph prompt of the predicted task is incorporated into the test graph for classification with the GNN. Since the graph prompts are learned task by task, *no data replay* is required in our TPP model. Further, the graph prompts are task-specific, so we essentially have a separate classification model for each graph task, *i.e.*, no continual model updating, completely avoiding the forgetting problem (*i.e.*, *forget-free*).

6.2 Related Work

Prompt Learning. Originating from natural language processing, prompt learning aims to facilitate the adaptation of frozen large-scale pre-trained models to

various downstream tasks by introducing learnable prompts [103]. In other words, prompt learning designs task-specific prompts to instruct the pre-trained models to perform downstream tasks conditionally. The prompts capture the knowledge of the corresponding tasks and enhance the compatibility between inputs and pre-trained models. Recently, prompt-based graph learning methods have also been proposed [142], which aims to unify multiple graph tasks [104] or improve the transferability of graph models [143]. Due to the ability to leverage the strong representative capacity of the pre-trained model and learn the knowledge of tasks in prompts, many prompting-based continual learning methods have been proposed [13,144,145] and achieved remarkable success without employing replaying memory or regularization terms. Despite that, no work is done on prompt learning for GCIL. The main challenge is the lack of pre-trained GNN models for all tasks in GCIL and the absence of task IDs to retrieve corresponding prompts during testing. In this chapter, we show that effective graph prompts can be learned for different tasks using a GNN backbone trained based on the first task with graph contrastive learning [113, 146], and our task ID prediction method and the graph prompts can be synthesized to address the GCIL problem.

6.3 Methodology

6.3.1 The GCIL Problem

Formally, GCL can be formulated as learning a model on a sequence of connected graphs (tasks) $\{\mathcal{G}^1, \ldots, \mathcal{G}^T\}$ where T is the number of tasks. Each $\mathcal{G}^t = (A^t, X^t)$ is a newly emerging graph, where A^t denotes the relations between N nodes of the current/new task, $X^t \in \mathbb{R}^{N \times F}$ represents the node attributes with dimensionality of F, and the labels of nodes can be denoted as Y^t. Each task contains a unique set of classes in a graph, *i.e.*,$\{Y^t \cap Y^j = \varnothing | t \neq j\}$. When learning task t, the model trained from previous tasks only has access to the current task data $\mathcal{G}^t$. The

goal is to accommodate the model to the current graph $\mathcal{G}^t$ while maintaining the classification performance on the previous graphs $\{\mathcal{G}^1, \ldots, \mathcal{G}^{t-1}\}$. In GCIL, the task IDs are not available during inference. Thus, assuming that each task has C classes, after learning all tasks, a GCIL model is required to classify a test instance into one of all the $T \times C$ classes.

6.3.2 Overview of The Proposed TPP Approach

Inspired by prior studies [14, 15], we decompose the class probability of a test sample $\mathbf{x}^{\text{test}}$ belonging to the j-th class in task t in GCIL into two parts :

$$H(y_j^t|\mathbf{x}^{\text{test}}) = H(y_j^t|\mathbf{x}^{\text{test}}, t)H(t|\mathbf{x}^{\text{test}})\,, \tag{6.1}$$

where $H(t|\mathbf{x}^{\text{test}})$ represents the task ID prediction probability of task t and $H(y_j^t|\mathbf{x}^{\text{test}}, t)$ denotes the prediction within the task t. This indicates that accurate GCIL classification accuracy can be achieved when both accurate task ID prediction and intra-task class classification are achieved.

To this end, in this chapter, we propose the Task Profiling and Prompting (**TPP**) approach for GCIL. As shown in Figure 6.2, a novel Laplacian smoothing-based task profiling approach is first devised in TPP for graph task ID prediction, which can well guarantee the task prediction accuracy as we will demonstrate theoretically below. Moreover, to obtain accurate intra-task class classification within the identified task, a novel graph prompting approach is further proposed to learn a small prompt for each task using a frozen GNN pre-trained on the first graph task. By learning and storing task knowledge separately, there is no knowledge interference between tasks during training, resulting in a model being both replay-free and forget-free. During inference, given a test sample, TPP first performs the task ID prediction and then retrieves the corresponding task graph prompt to concatenate with the sample for the GCIL classification. Below we introduce the TPP approach in detail.

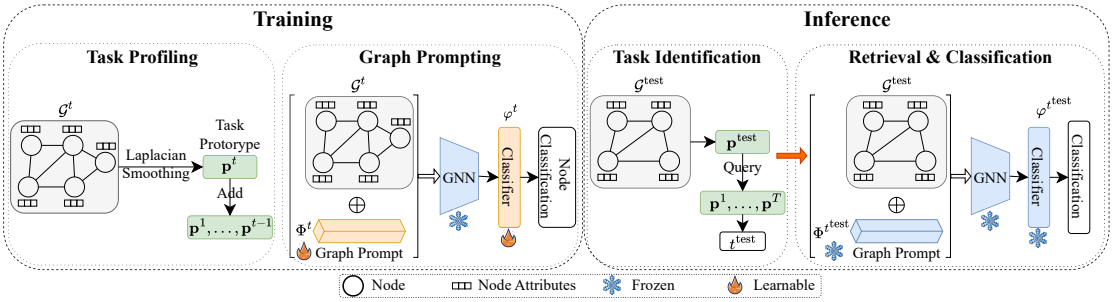


Figure 6.2 : Overview of the proposed TPP approach. During training, for each graph task t, the task prototype $\mathbf{p}^t$ is generated by applying Laplacian smoothing on the graph $\mathcal{G}^t$ and added to $\mathcal{P} = \{\mathbf{p}^1, \ldots, \mathbf{p}^{t-1}\}$. At the same time, the graph prompt Φ^t and the classification head φ^t for this task are optimized on $\mathcal{G}^t$ through a frozen pre-trained GNN. During inference, the task ID of the test graph is first inferred (*i.e.*, task identification). Then, the graph prompt and the classifier of the predicted task are retrieved to perform the node classification in GCIL. The GNN is trained on $\mathcal{G}^1$ and remains frozen for subsequent tasks.

6.3.3 Laplacian Smoothing-based Task Profiling for Graph Task ID Prediction

To leverage the graph structure and node attribute information, we propose to use a Laplacian smoothing approach to generate a prototypical embedding for each graph task for task ID prediction. Specifically, for the task t with graph data $\mathcal{G}^t = (A^t, X^t)$, we construct a task prototype $\mathbf{p}^t$ for this task based on the train set denoted as $\{x_i | i \in \mathcal{V}^t_{\text{train}}\}$, where $\mathcal{V}^t_{\text{train}}$ is the train set of $\mathcal{G}^t$. Given $\mathcal{G}^t$, the Laplacian smoothing is first applied on the graph $\mathcal{G}_t$ to obtain the smoothed node embeddings Z^t:

$$Z^t = (I - (\hat{D}^t)^{-\frac{1}{2}} \hat{L}^t (\hat{D}^t)^{-\frac{1}{2}})^s X^t \,, \tag{6.2}$$

where s denotes the number of Laplacian smoothing steps, I is an identity matrix, and $\hat{L}^t$ is the graph Laplacian [147] matrix of $\hat{A}^t = A^t + I$ (*i.e.*, $\hat{L}^t = \hat{D}^t - \hat{A}^t$ with $\hat{D}^t$

being the diagonal degree matrix of $\hat{A}^t$ and $\hat{D}^t_{ii} = \sum_j \hat{a}_{ij}$). Then, the task prototype $\mathbf{p}^t$ is constructed by averaging the smoothed embeddings of train nodes:

$$\mathbf{p}^t = \frac{1}{|\mathcal{V}^t_{\text{train}}|} \sum_{i \in \mathcal{V}^t_{\text{train}}} \mathbf{z}^t_i (\hat{D}^t_{ii})^{-\frac{1}{2}} . \tag{6.3}$$

Similarly, the task prototypes for all tasks can be separately constructed and stored, denoted as $\mathcal{P} = \{\mathbf{p}^1, \ldots, \mathbf{p}^T\}$. Given a test graph $\mathcal{G}^{\text{test}}$ at testing time, we predict the task ID of $\mathcal{G}^{\text{test}}$ by querying the task prototype pool $\mathcal{P}$. Specifically, the task prototype of $\mathcal{G}^{\text{test}}$ is obtained with the set of test nodes in a similar way as on training graphs via Eq. (6.2) and Eq. (6.3), *i.e.*,

$$\mathbf{p}^{\text{test}} = \frac{1}{|\mathcal{V}^{\text{test}}|} \sum_{i \in \mathcal{V}^{\text{test}}} \mathbf{z}^{\text{test}}_i (\hat{D}^{\text{test}}_{ii})^{-\frac{1}{2}} , \tag{6.4}$$

where $\mathcal{V}^{\text{test}}$ denotes the set of nodes to be classified in $\mathcal{G}^{\text{test}}$ and $\mathbf{z}^{\text{test}}_i$ is the smoothed embedding of the test node i after s-step Laplacian smoothing. Then, we query the task prototype pool $\mathcal{P}$ with the test prototype $\mathbf{p}^{\text{test}}$ and return the task ID whose task prototype is most similar to $\mathbf{p}^{\text{test}}$:

$$t^{\text{test}} = \arg\min(d(\mathbf{p}^{\text{test}}, \mathbf{p}^1), \ldots, d(\mathbf{p}^{\text{test}}, \mathbf{p}^T))) , \tag{6.5}$$

where $d(\cdot)$ represents an Euclidean distance function and t^{test} is the predicted task ID of $\mathcal{G}^{\text{test}}$.

As discussed in Sec. 6.3.2, more accurate task ID prediction leads to better classification performance for GCIL. Below we show theoretically that the task ID of the test graphs can be accurately predicted with our simple Laplacian smoothing-based task profiling approach.

Theorem 1. *If graphs for all tasks are not isolated and the test graph $\mathcal{G}^{test}$ comes from the task t, i.e., $\mathcal{G}^{test}$ and $\mathcal{G}^t$ have the same set of classes, then the distance between $\mathbf{p}^{test}$ and $\mathbf{p}^t$ approaches to zero with a sufficiently large number of Laplacian smoothing steps s:*

$$\lim_{s \to +\infty} d(\mathbf{p}^{test}, \mathbf{p}^t) = 0 . \tag{6.6}$$

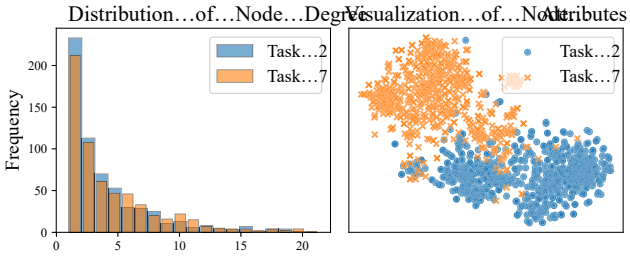


Figure 6.3 : The differences between two graphs in structure and node attributes.

Theorem 2. *Suppose the test graph comes from task t, and let* $\mathbf{e}$ *and* ϵ *be the differences in node degrees and node attributes between two different tasks t and j respectively, which are defined by* $(\hat{D}^j)^{\frac{1}{2}} = (\hat{D}^t)^{\frac{1}{2}} + \mathrm{Diag}(\mathbf{e})$ *and* $X^j = X^t + \epsilon$. *Then the distance between the task prototypes of task t and j obtained with large steps of Laplacian smoothing can be explicitly calculated as:*

$$d(\mathbf{p}^{test}, \mathbf{p}^j) = \|(\mathbf{e}_N^t)^T \epsilon + (\mathbf{e})^T X^j\|_2 \,, \tag{6.7}$$

where $\mathbf{e}_N^t = (\hat{D}^t)^{\frac{1}{2}}[1, 1, \ldots, 1]^T$ *is the N-th eigenvector of task t and* $(\mathbf{e}_N^t)^T$ *denotes its transpose.*

The two theorems indicate that i) if the test graph belongs to task t, with a large s, the distance between $\mathbf{p}^{\text{test}}$ and $\mathbf{p}^t$ would become zero with the proposed Laplacian smoothing and prototype construction method (Theorem 1); and ii) for graphs from different tasks, since they contain different set of classes, they have large differences in graph structure and node attributes, which can lead to a large distance between task prototypes $\mathbf{p}^t$ and $\mathbf{p}^j$ (Theorem 2), thereby having the following inequality hold if $\mathcal{G}^{\text{test}}$ comes from task t:

$$d(\mathbf{p}^{\text{test}}, \mathbf{p}^t) < d(\mathbf{p}^{\text{test}}, \mathbf{p}^j) \,. \quad \forall j \neq t \,. \tag{6.8}$$

Without loss of generality, we empirically investigate the differences between two randomly chosen graph tasks of the CorFull dataset in Figure 6.3. We can see that

the two graphs of the tasks have a rather large difference in both graph structure and node attributes. The larger the differences in $\mathbf{e}$ and ϵ of the two graphs, the larger the gap is between $d(\mathbf{p}^{\text{test}}, \mathbf{p}^t)$ and $d(\mathbf{p}^{\text{test}}, \mathbf{p}^j)$. As a result, the task of the test graph can be predicted accurately with Eq. (6.5). In the experimental section, we empirically evaluate the proposed task ID prediction and report its accuracy on different datasets.

Proof of Theorems

Proof of Theorem 1.

Proof. To prove the distance between $\mathbf{p}^{\text{test}}$ and $\mathbf{p}^t$ approaches 0 with a large Laplacian smoothing step s, we need to illustrate that the features of nodes in $\mathcal{G}^t$ coverage to be proportional to the square root of the node degree after Laplacian smoothing and the proposed task prototype construction method transforms all node features to the same values. Note that the self-loop is added to each node in the graph, resulting in the graph being non-bipartile. Assuming the size of the graph $\mathcal{G}^t$ is N, the Laplacian matrix $(\hat{D}^t)^{-\frac{1}{2}}\hat{L}^t(\hat{D}^t)^{-\frac{1}{2}}$ has N eigenvalues with different eigenvectors [148]. Recalling the Laplacian smoothing defined in Eq. (6.2), the eigenvalues and eigenvectors of $I - (\hat{D}^t)^{-\frac{1}{2}}\hat{L}^t(\hat{D}^t)^{-\frac{1}{2}}$ can be represented as $(\lambda_1, \ldots, \lambda_N)$ and $(\mathbf{e}_1, \ldots, \mathbf{e}_N)$ respectively. With the property of symmetric Laplacian matrix for the non-bipartile graph, the eigenvalues of $I - (\hat{D}^t)^{-\frac{1}{2}}\hat{L}^t(\hat{D}^t)^{-\frac{1}{2}}$ are all in the range of $(-1, 1]$ [147], *i.e.*,

$$-1 < \lambda_1 < \ldots < \lambda_N = 1\,, \quad \mathbf{e}_N = (\hat{D}^t)^{\frac{1}{2}}[1, 1, \ldots, 1]^T \in \mathbb{R}^{N\times 1}\,. \tag{6.9}$$

Based on the eigenvalues and the eigenvectors, the result of applying Laplacian smoothing on the node features X^t after s steps can be formulated as:

$$(I - (\hat{D}^t)^{-\frac{1}{2}}\hat{L}^t(\hat{D}^t)^{-\frac{1}{2}})^s X^t = [\lambda_1^s \mathbf{e}_1, \ldots, \lambda_N^s \mathbf{e}_N]\hat{X}^t\,, \tag{6.10}$$

where $\hat{X}^t = [\mathbf{e}_1, \ldots, \mathbf{e}_N]^{-1} X^t$. As the eigenvectors are orthogonal to each other, we can further rewrite $\hat{X}^t$ as $\hat{X}^t = [\mathbf{e}_1, \ldots, \mathbf{e}_N]^T X^t$. Since the absolute values of the eigenvalues are less than 1 except λ_N, λ_i^s would approach 0 as s go infinity, *i.e.*, $\lim_{s\to\infty} \lambda_i^s = 0, \forall i \neq N$. Then, we can formulate the smoothed node feature representations Z^t as follows:

$$Z^t = [(\hat{D}_{11}^t)^{\frac{1}{2}}, \ldots, (\hat{D}_{NN}^t)^{\frac{1}{2}}]^T \hat{X}^t[N, :]\,, \tag{6.11}$$

where $\hat{X}^t[N, :]$ denotes the N-th row of $\hat{X}^t$. Therefore, with a larger s, the feature of nodes in $\mathcal{G}^t$ would converge to be proportional to the square root of the node degree. By multiplying the smoothed feature with $(\hat{D}_{ii}^t)^{-\frac{1}{2}}$ for each node i in the proposed task prototype construction, $\mathbf{p}^{\text{test}}$ and $\mathbf{p}^t$ would both become to be $\hat{X}^t[N, :]$, despite that they utilize different nodes to construct the task prototypes. Therefore, the distance between $\mathbf{p}^{\text{test}}$ and $\mathbf{p}^t$ would become zero with a large s. □

Note that we assume that the graphs of all tasks are not isolated in Theorem 1. Having isolated nodes in real-world graphs would deviate from this assumption, resulting in unsatisfactory task identification. To tackle this issue, we propose a simple graph augmentation when constructing the task prototypes, which adds an edge between the isolated nodes and the randomly chosen non-isolated nodes to make the graph more connected. This helps the proposed task ID prediction method predict the task of test graphs more accurately.

Proof of Theorem 2.

Proof. As derived in Theorem 1, the task prototypes of task t and j with large steps of Laplacian smoothing are $\mathbf{p}^{\text{test}} = \hat{X}^t[N, :]$ and $\mathbf{p}^j = \hat{X}^j[N, :]$ respectively. Furthermore, the task prototype of task t can be represented as $\mathbf{p}^t = (\mathbf{e}_N^t)^T X^t$. Based on the difference in node degrees and node attributes between task t and j,

the distance between the task prototypes can be represented as follows:

$$d(\mathbf{p}^{\text{test}}, \mathbf{p}^j) = \|\mathbf{p}^t - \mathbf{p}^j\|_2 \tag{6.12}$$

$$= \|(\mathbf{e}_N^t)^T X^t - (\mathbf{e}_N^j)^T X^j\|_2 \tag{6.13}$$

$$= \|(\mathbf{e}_N^t)^T X^t - (\mathbf{e}_N^t + \mathbf{e})^T (X^t + \epsilon)\|_2 \tag{6.14}$$

$$= \|(\mathbf{e}_N^t)^T \epsilon + (\mathbf{e})^T X^j\|_2 \tag{6.15}$$

□

6.3.4 Graph Prompt Learning for GCIL

Instead of utilizing regularization or replaying memory as in existing GCL methods, TPP aims to learn a task-specific prompt for each graph task. The information of each graph task can be explicitly modeled and stored in a separate task-specific graph prompt, with the GNN backbone being frozen. This effectively avoids the forgetting of knowledge of any previous tasks and the interference between tasks. To this end, the graph prompt in TPP is designed as a set of learnable tokens that can be incorporated into the feature space of the graph data for each task. Specifically, for a task t, the graph prompt can be represented as $\Phi^t = [\phi_1^t, \ldots, \phi_k^t]^T \in \mathbb{R}^{k \times F}$ where k is the number of vector-based tokens ϕ^i. For each node in $\mathcal{G}^t$, the node attribute is augmented by the weighted combination of these tokens, with the weights obtained from k learnable linear projections:

$$\bar{\mathbf{x}}_i^t = \mathbf{x}_i^t + \sum_j^k \alpha_j \phi_j^t\,, \quad \alpha_j = \frac{e^{(\mathbf{w}_j)^T \mathbf{x}_i^t}}{\sum_l^k e^{(\mathbf{w}_l)^T \mathbf{x}_i^t}}\,, \tag{6.16}$$

where α_j denotes the importance score of the token ϕ^j in the prompt and $\mathbf{w}_j$ is a learnable projection. For convenience, we denote the graph modified by the graph prompt as $\bar{\mathcal{G}}^t = (A^t, X^t + \Phi^t)$. Then, $\bar{\mathcal{G}}^t$ is fed into a frozen pre-trained GNN model $f(\cdot)$ to obtain the embeddings for classification. In TPP, we employ a single-layer MLP as the classifier attached to the GNN, denoting φ^t for task t. The node

classification at task t can be formulated as:

$$\hat{Y}^t = \varphi^t(f(A^t, X^t + \Phi^t)). \tag{6.17}$$

Therefore, the graph prompt and the MLP-based classification head are optimized by minimizing a node classification loss:

$$\min_{\Phi^t, \varphi^t} \frac{1}{|\mathcal{V}^t_{\text{train}}|} \sum_{i \in \mathcal{V}^t_{\text{train}}} \ell_{\text{CE}}(\hat{y}^t_i, y^t_i)\,, \tag{6.18}$$

where $\mathcal{V}^t_{\text{train}}$ is the train set of $\mathcal{G}^t$, y^t_i is the label of node i, $\hat{y}^t_i \in \hat{Y}^t$ is the predicted label, and $\ell_{\text{CE}(\cdot)}$ is a cross-entropy loss. By minimizing Eq. (6.16), the graph prompt and the classifier are learned to leverage the generic, cross-task knowledge in the frozen GNN $f(\cdot)$ for the task t. Meanwhile, Φ^t and φ^t learn specific knowledge for the task t. This essentially results in a separate classification model for each task, and no data replay is required for all tasks. As a result, TPP is fully free of catastrophic forgetting for GCIL. An alternative approach to overcoming forgetting is to learn a separate GNN model for each task. However, this would introduce heavy burdens on optimization and storage with the increasing number of tasks. By contrast, the proposed graph prompting only introduces minimal parameters for each task as the prompts are very small.

Details on Learning GNN Backbone

We propose to construct a GNN backbone $f(\cdot)$ for graph prompt learning so that the task-specific information can be absorbed into the prompts. Specifically, the model $f(\cdot)$ is constructed based on the first task $\mathcal{G}^1 = (A^1, X^1)$ via graph contrastive learning due to its ability to obtain transferable models [113, 146] across graphs.

To construct contrastive views for graph contrastive learning, two widely used graph augmentations are employed, *i.e.*, edge removal and attribute masking [113].

Specifically, the edge removal randomly drops a certain portion of existing edges in $\mathcal{G}^1$ and the attribute masking randomly masks a fraction of dimensions with zeros in node attributes, *i.e.*,

$$\tilde{A}^1 = A^1 \circ R\,, \quad \tilde{X}^1 = [\mathbf{x}_1^1 \circ \mathbf{m}, \ldots, \mathbf{x}_N^1 \circ \mathbf{m}]^T\,, \tag{6.19}$$

where $R \in \{0,1\}^{N\times N}$ is the edge masking matrix whose entry is drawn from a Bernoulli distribution controlled by the edge removal probability, $\mathbf{m} \in \{0,1\}^F$ is the attribute masking vector whose entry is independently drawn from a Bernoulli distribution with the attribute masking ratio, and $\circ$ denotes the Hadamard product. By applying the graph augmentations to the original graph, the corrupted graph $\tilde{\mathcal{G}}^1 = (\tilde{A}^1, \tilde{X}^1)$ forms the contrastive view for the original graph $\mathcal{G}^1 = (A^1, X^1)$. Then, $\tilde{\mathcal{G}}^1$ and $\mathcal{G}^1$ are inputted to the shared GNN $f(\cdot)$ followed by non-linear projection $g(\cdot)$ to obtain the corresponding node embeddings, *i.e.*, $\tilde{Z}^1 = g(f(\tilde{\mathcal{G}}^1))$ and $Z^1 = g(f(\mathcal{G}^1))$. For graph contrastive learning, the embeddings of the same node in different views are pulled closer while the embeddings of other nodes are pushed apart. The pairwise objective for each node pair $(\tilde{\mathbf{z}}_i^1, \mathbf{z}_i^1)$ can be formulated as:

$$\ell(\tilde{\mathbf{z}}_i^1, \mathbf{z}_i^1) = -\log \frac{e^{sim(\tilde{\mathbf{z}}_i^1, \mathbf{z}_i^1)/\tau}}{e^{sim(\tilde{\mathbf{z}}_i^1, \mathbf{z}_i^1)/\tau} + \sum_{j\neq i}^N e^{sim(\tilde{\mathbf{z}}_i^1, \mathbf{z}_j^1)/\tau} + \sum_{j\neq i}^N e^{sim(\tilde{\mathbf{z}}_i^1, \tilde{\mathbf{z}}_j^1)/\tau}}\,, \tag{6.20}$$

where $sim(\cdot)$ represents the cosine similarity and τ is a temperature hyperparameter. Therefore, the overall objective can be defined as follows:

$$\mathcal{L}_{\text{contrast}} = \frac{1}{2N}\sum_{i=1}^{N}(\ell(\tilde{\mathbf{z}}_i^1, \mathbf{z}_i^1) + \ell(\mathbf{z}_i^1, \tilde{\mathbf{z}}_i^1))\,. \tag{6.21}$$

With the objective Eq. (6.21), the model $f(\cdot)$ is optimized to learn discriminative representations of nodes. Despite the limited size of the first task, the learned model $f(\cdot)$ can effectively adapt to other tasks with the proposed graph prompt learning method.

6.3.5 Training and Inference in TPP

Training. The training of TPP can be divided into two parts. First, for each task $\mathcal{G}^t$, the prototypical embedding $\mathbf{p}^t$ is generated based on Laplacian smoothing and stored in $\mathcal{P}$ for task ID prediction. Then, the information of $\mathcal{G}^t$ is explicitly modeled and stored with the proposed graph prompt learning, *i.e.*, Eq. (6.18). For the GNN backbone $f(\cdot)$ in graph prompt learning, we propose to learn it based on the first task $\mathcal{G}^1 = (A^1, X^1)$ via graph contrastive learning due to its ability to obtain transferable models [113, 146] across graphs. Despite being only learned on $\mathcal{G}^1$, $f(\cdot)$ can effectively adapt to all subsequent tasks with graph prompts. Overall, after learning all tasks in $\{\mathcal{G}^1, \ldots, \mathcal{G}^T\}$, the task profiles and task-specific information are explicitly modeled in $\mathcal{P} = \{\mathbf{p}^1, \ldots, \mathbf{p}^T\}$, $\{\Phi^1, \ldots, \Phi^T\}$ and $\{\varphi^1, \ldots, \varphi^T\}$.

Inference. Given the test graph $\mathcal{G}^{\text{test}}$, the task prototype $\mathbf{p}^{\text{test}}$ is constructed with Eq. (6.4) and then used to obtain the task ID t^{test} by querying $\mathcal{P} = \{\mathbf{p}^1, \ldots, \mathbf{p}^T\}$, *i.e.*, via Eq. (6.5). Finally, the test graph $\mathcal{G}^{\text{test}}$ is augmented with the corresponding graph prompt $\Phi^{t^{\text{test}}}$ and fed into the GNN and the classification head constructed with $f(\cdot)$ and $\varphi^{t^{\text{test}}}$ respectively to get the node classification results. Formally, the inference can be formulated as:

$$\begin{cases} t^{\text{test}} = \arg\min(d(\mathbf{p}^{\text{test}}, \mathbf{p}^1), \ldots, d(\mathbf{p}^{\text{test}}, \mathbf{p}^T))\,, \\ Y^{\text{test}} = \varphi^{t^{\text{test}}}(f(A^{\text{test}}, X^{\text{test}} + \Phi^{t^{\text{test}}}))\,. \end{cases} \tag{6.22}$$

The training and inference processes of the proposed method are summarized in Algorithm 5 and Algorithm 6, respectively.

6.3.6 Time Complexity Analysis

The proposed method first learns a GNN backbone based on the first task with graph contrastive learning. Then, the model remains frozen and the task-specific graph prompts and classifiers are optimized to capture the knowledge of

Algorithm 5 Training of TPP

Input: A series of graph learning tasks: $\{\mathcal{G}^1, \ldots, \mathcal{G}^T\}$, a graph neural network $f(\cdot)$.
Output: Graph neural network $f(\cdot)$, task prototype $\mathcal{P} = \{\mathbf{p}^1, \ldots, \mathbf{p}^T\}$, graph prompts $\{\Phi^1, \ldots, \Phi^T\}$, and classifiers $\{\varphi^1, \ldots, \varphi^T\}$.
Pre-train $f(\cdot)$ on $\mathcal{G}^1$ with graph contrastive learning (Eq. (6.21)).
for $t = 1, \ldots, T$ **do**
    Obtain the task prototype $\mathbf{p}^t$ of task t (Eq. (6.3)).
    Obtain $\bar{\mathcal{G}}^t$ with graph prompts Φ^t (Eq. (6.16)).
    Optimize Φ^t and φ^t by minimizing node classification loss (Eq. (6.18)).
end for

Algorithm 6 Inference in TPP

Input: Graph neural network $f(\cdot)$, task prototypes $\mathcal{P} = \{\mathbf{p}^1, \ldots, \mathbf{p}^T\}$, graph prompts $\{\Phi^1, \ldots, \Phi^T\}$, classifiers $\{\varphi^1, \ldots, \varphi^T\}$, and the test graph $\mathcal{G}^{\text{test}}$.
Output: Prediction result.
Obtain the task prototype $\mathbf{p}^{\text{test}}$ (Eq. (6.4)).
Infer the task t^{test} of the test graph by querying $\mathcal{P}$ with $\mathbf{p}^{\text{test}}$ (Eq. (6.5)).
Retrieve the corresponding graph prompt $\Phi^{t^{\text{test}}}$ and classifier $\varphi^{t^{\text{test}}}$
Obtain $\bar{\mathcal{G}}^{\text{test}} = (A^{\text{test}}, X^{\text{test}} + \Phi^{t^{\text{test}}})$ (Eq. (6.16)).
return $\varphi^{t^{\text{test}}}(f(A^{\text{test}}, X^{\text{test}} + \Phi^{t^{\text{test}}}))$

each task separately. In experiments, we employ a two-layer SGC [112] as the GNN backbone model with the number of hidden units in all layers as d. Suppose all tasks contain the same number of nodes as N, the time complexity of the graph contrastive learning on the first task is $\mathcal{O}((4|A^1|F + 2NdF + 3Nd^2)E_1)$, where $|A^1|$ returns of the number of edges of the $\mathcal{G}^1$, F represents the dimensionality of node attributes and E_1 is the number of training epochs. After that, we propose to freeze the learned model and learn graph prompts and classifiers for each task. In our experiments, we set the size of each graph prompt to k

and implement the classification head as a single-layer MLP outputting the probabilities of C classes. Given the number of training epochs E_2, the time complexity of optimizing the graph prompt and classifier is $\mathcal{O}((4kNF + 2dNC)E_2)$, which includes both the forward and backward propagation. Despite the graph model being frozen, the forward and backward propagation of the model are still needed to optimize the task-specific graph prompts and classifiers. Given the number of tasks T in GCIL, the overall time complexity of the proposed method is $\mathcal{O}((4|A^1|F+2NdF+3Nd^2)E_1+\sum_{i=1}^{T}(4|A^1|F+2NdF+3Nd^2+4kNF+2dNC)E_2)$, which is linear to the number of nodes, the number of edges, and the number of node attributes involved in all the graph tasks.

6.4 Experiments

Datasets. Following the GCL performance benchmark [10], four large public graph datasets are employed, including CoraFull [135], Arxiv [136], Reddit [5] and Products [136]. Specifically, CoraFull and Arxiv are citation networks, Reddit is constructed from Reddit posts, and Products is a co-purchasing network from Amazon. For all datasets, each task is set to contain only two classes [10]. Besides, for each class, the proportions of training, validation, and testing are set to be 0.6, 0.2, and 0.2, respectively.

Baseline Methods. Two categories of state-of-the-art (SOTA) continual learning methods are employed for comparison: (1) general CIL methods: EWC [125], LwF [138], GEM [139] and MAS [140]; (2) graph CIL methods: ERGNN [54], TWP [55], SSM [61], SEM [63], CaT [64] and DeLoMe [65]. There are limited methods on task ID prediction for CIL, *e.g.*, [14, 16], but they are not suited for graph data. To compare with this type of methods, we adapt the OOD detection-based CIL methods [14, 16] for GCIL (named OODCIL). In addition, we include two baseline methods: **Fine-Tune** and **Joint**. The Fine-Tune method is a baseline that simply fine-tunes

the learned model from previous tasks without continual learning techniques, while the Joint method is an oracle model that can see all graphs at all times and performs GCL on the full graphs of all tasks. We also report the results of an enhanced **Oracle Model** that is an enhanced version of the Joint method with access to the task ID of every test sample during inference. The details of these methods are listed as follows:

- **EWC** [125] is a regularization-based method that adds a quadratic penalty on the model parameters according to their importance to the previous tasks to maintain the performance on previous tasks.
- **MAS** [140] preserves the parameters important to previous tasks based on the sensitivity of the predictions to the changes in the parameters.
- **GEM** [139] stores representative data in the episodic memory and proposes to modify the gradients of the current task with the gradient calculated on the memory data to tackle the forgetting problem.
- **LwF** [138] employs knowledge distillation to minimize the discrepancy between the logits of the old and the new models to preserve the knowledge of the previous tasks.
- **TWP** [55] proposes to preserve the important parameters in the topological aggregation and loss minimization for previous tasks via regularization terms.
- **ERGNN** [54] is a replay-based method that constructs memory data by storing representative nodes selected from previous tasks.
- **SSM** [61] incorporates the explicit topological information of selected nodes in the form of sparsified computation subgraphs into the memory for graph continual learning.

- **SEM** [63] improves SSM by storing the most informative topological information via the Ricci curvature-based graph sparsification technique.

- **CaT** [64] condenses each graph to a small synthesized replayed graph and stores it in a condensed graph memory with historical replay graphs. Moreover, graph continual learning is accomplished by updating the model directly with the condensed graph memory.

- **DeLoMe** [65] learns lossless prototypical node representations as the memory to capture the holistic graph information of previous tasks. A debiased GCL loss function is further devised to address the data imbalance between the classes in the memory data and the current data.

Details on the Design of the OODCIL Method We adapt the OOD detection-based CIL methods in [14, 16] for empirical comparison under GCIL. To this end, following [14,16], we propose to build an OOD detector for each graph task to perform task ID prediction and within-task classification simultaneously. Specifically, for task t with C classes, we aim to learn an OOD detector $f_o^t(\cdot)$ with $C+1$ classes. The extra class represents the OOD data for this task. In this chapter, we implement the OOD detector as a two-layer SGC [112] model and take the data in replay buffer $Buf_{<t}$ as the OOD data for task t. Formally, the OOD detector is optimized by minimizing the following objective for task t:

$$\min_{f_o^t(\cdot)} \ \mathbb{E}_{\mathcal{G}^t \bigcup Buf_{<t}} \left[\ell_{\text{CE}}(f_o^t(\mathcal{G}^t), Y^t) + \ell_{\text{CE}}(f_o^t(Buf_{<t}), Y^{Buf}) \right] , \tag{6.23}$$

where $Y^{Buf} = C+1$ represents the labels of data in the replay buffer $Buf_{<t}$. The buffer $Buf_{<t}$ is constructed via sampling previous graphs following ERGNN [54]. Note that there are no replay data for task 1 to train the OOD detector. To overcome this issue, we propose to generate OOD data for task 1 based on graph augmentation [113].

After learning all the tasks, we follow Eq. (6.1) to compute the class probabilities for the test samples. Specifically, a test sample is fed into all the learned OOD detectors to obtain the OOD score and class probabilities within the corresponding tasks. For example, for task t, the learned OOD detector would output the class and OOD probabilities $\{y_1^t, \ldots, y_C^t, o^t\}$ for the test sample. As the OOD score indicates the probability of the test sample is OOD to this task, the final probabilities of the test sample w.r.t. the classes in task t can be calculated as $\{(1-o^t)y_1^t, \ldots, (1-o^t)y_C^t\}$. Among all the class probabilities of all tasks, the class with the highest probability is predicted as the class for the test sample.

Implementation Details. All the continual learning methods, including the proposed method, are implemented based on the GCL benchmark [10]*. As in [63], we employ a two-layer SGC [112] model as the backbone model with the same hyperparameters as [63]. The hidden dimension is set to 256 for all methods. The number of training epochs of each graph learning task is 200, with Adam as the optimizer, and the learning rate is set to 0.005 by default. For task prototype construction, the number of steps s in Laplacian smoothing is set to 3 by default. The number of tokens in each graph prompt, k, is also set to 3 across the four datasets. For the memory-replay methods, we follow the settings in [65].

For graph contrastive learning, the probabilities of edge removal and attribute masking are set to 0.2 and 0.3 respectively for all datasets. Besides, the learning rate is set to 0.001 with Adam optimizer, the training epochs are set to 200 and the temperature τ is 0.5 for all datasets.

The code is implemented with Pytorch (version: 1.10.0), DGL (version: 0.9.1), OGB (version: 1.3.6), and Python 3.8.5. Besides, all experiments are conducted on a Linux server with an Intel CPU (Intel Xeon E-2288G 3.7GHz) and an Nvidia

*`https://github.com/QueuQ/CGLB/tree/master`

GPU (Quadro RTX 6000). For each dataset, we report the average performance with standard deviations after 5 independent runs with different random seeds.

Evaluation Metrics. To evaluate the performance of continual learning methods, two commonly used metrics, average accuracy (AA) and average forgetting (AF) after all tasks have been learned, are adopted in our experiments. Specifically, average accuracy (AA) and average forgetting (AF) are calculated from the lower diagonal accuracy matrix $M \in \mathbb{R}^{T\times T}$, where T is the number of the tasks. The entry $M_{tj}(t \geqslant j)$ denotes the classification accuracy on task j after the model is optimized on task t. Therefore, the row in $M_{t,:}$ records the performance on all previous tasks after learning task t, and the column $M_{:,j}$ describes the dynamic of performance on task j when learning different tasks. After learning all the T tasks, the overall average accuracy (AA) and average forgetting (AF) can be calculated as follows:

$$\mathrm{AA} = \frac{\sum_{j=1}^{T} M_{Tj}}{T}, \quad \mathrm{AF} = \frac{\sum_{j=1}^{T-1}(M_{Tj} - M_{jj})}{T-1}. \tag{6.24}$$

To sum up, AA evaluates the average performance of the model on all the learned tasks after learning all the T tasks, and AF describes how the performance of previous tasks is affected when learning the current task. A positive AF indicates that learning the current task would facilitate the previous tasks and vice versa. For both AA and AF, the higher value denotes better GCL performance.

6.4.1 Main Results

Comparison to SOTA Methods. The results of TPP and its baseline methods under the GCIL setting are shown in Table 6.1. From the table, we can draw the following key observations. (1) As demonstrated by the results of Fine-Tune, directly fine-tuning the learned model from previous tasks on the current task data leads to serious performance degradation because the knowledge of previous tasks could be easily overwritten by the new tasks. (2) CIL methods proposed for Euclidean data generally do not achieve satisfactory performance for GCIL, which verifies the fact

Table 6.1 : Results (mean±std) under the GCIL setting on four large datasets. The best performance on each dataset is boldfaced. "↑" denotes the higher value represents better performance. Oracle Model can get access to the data of all tasks and task IDs, *i.e.*, it obtains the upper bound performance. "✓" in Data Replay indicates the use of data replay in the model, and × denotes no data replay involved.

Methods	Data Replay	CoraFull		Arxiv		Reddit		Products	
		AA/%↑	AF/%↑	AA/%↑	AF/%↑	AA/%↑	AF/%↑	AA/%↑	AF/%↑
Fine-tune	×	3.5±0.5	-95.2±0.5	4.9±0.0	-89.7±0.4	5.9±1.2	-97.9±3.3	7.6±0.7	-88.7±0.8
Joint	×	81.2±0.4	-	51.3±0.5	-	97.1±0.1	-	71.5±0.1	-
EWC	×	52.6±8.2	-38.5±12.1	8.5±1.0	-69.5±8.0	10.3±11.6	-33.2±26.1	23.8±3.8	-21.7±7.5
MAS	×	6.5±1.5	-92.3±1.5	4.8±0.4	-72.2±4.1	9.2±14.5	-23.1±28.2	16.7±4.8	-57.0±31.9
GEM	×	8.4±1.1	-88.4±1.4	4.9±0.0	-89.8±0.3	11.5±5.5	-92.4±5.9	4.5±1.3	-94.7±0.4
LwF	×	33.4±1.6	-59.6±2.2	9.9±12.1	-43.6±11.9	86.6±1.1	-9.2±1.1	48.2±1.6	-18.6±1.6
TWP	×	62.6±2.2	-30.6±4.3	6.7±1.5	-50.6±13.2	8.0±5.2	-18.8±9.0	14.1±4.0	-11.4±2.0
ERGNN	✓	34.5±4.4	-61.6±4.3	21.5±5.4	-70.0±5.5	82.7±0.4	-17.3±0.4	48.3±1.2	-45.7±1.3
SSM-uniform	✓	73.0±0.3	-14.8±0.5	47.1±0.5	-11.7±1.5	94.3±0.1	-1.4±0.1	62.0±1.6	-9.9±1.3
SSM-degree	✓	75.4±0.1	-9.7±0.0	48.3±0.5	-10.7±0.3	94.4±0.0	-1.3±0.0	63.3±0.1	-9.6±0.3
SEM-curvature	✓	77.7±0.8	-10.0±1.2	49.9±0.6	-8.4±1.3	96.3±0.1	-0.6±0.1	65.1±1.0	-9.5±0.8
CaT	✓	80.4±0.5	-5.3±0.4	48.2±0.4	-12.6±0.7	97.3±0.1	-0.4±0.0	70.3±0.9	-4.5±0.8
DeLoMe	✓	81.0±0.2	-3.3±0.3	50.6±0.3	5.1±0.4	97.4±0.1	-0.1±0.1	67.5±0.7	-17.3±0.3
OODCIL	✓	71.3±0.5	-1.1±0.1	19.3±1.4	-1.0±0.4	79.3±0.8	-0.1±0.0	41.6±0.9	-1.6±0.4
TPP (Ours)	×	**93.4±0.4**	**0.0±0.0**	**85.4±0.1**	**0.0±0.0**	**99.5±0.0**	**0.0±0.0**	**94.0±0.5**	**0.0±0.0**
Oracle Model	×	95.5±0.2	-	90.3±0.4	-	99.5±0.0	-	95.3±0.8	-

that the unique graph properties should be taken into consideration for GCIL. (3) Replay-based methods generally achieve much better performance than the other baselines, showing the effectiveness of using an external memory buffer to overcome catastrophic forgetting. However, all of them still suffer from forgetting, in addition to the inter-task separation issue. (4) The performance of OODCIL demonstrates that despite achieving impressive AF performance, current OOD detection-based CIL methods are not effective for GCIL due to the overlook of graph properties in its OOD detector and classification model. (5) Different from the baselines that involve the forgetting problem to varying extents, the proposed method TPP is

a fully forget-free GCIL approach, achieving an AF value of zero across all four datasets. TPP is also consistently the best performer in AA, outperforming the best baseline method by over 18% in AA averaged over the four datasets. This superiority is attributed to the highly accurate task ID prediction module in TPP and its effective task-specific graph prompt learning (see Sec. 6.4.2). (6) Our method lifts the SOTA AA performance by a large margin and even significantly outperforms the oracle baseline Joint in all cases. This is because although the Joint method can mitigate catastrophic forgetting due to its access to the data of all graphs, it is still challenged by the inter-task class separation issue since it is not given task ID during inference. TPP effectively tackles both catastrophic forgetting and inter-task class separation issues, thus achieving significantly better AA than Joint and very comparable AA to the Oracle Model.

Enabling Existing GCIL Methods with Our Task ID Prediction Module. Existing GCIL Methods often suffer from a severe inter-task class separation issue. Our task ID prediction is devised as a module to tackle this issue. To show its effectiveness as an individual plug-and-play module, we evaluate its performance when combined with existing GCIL methods. Our task ID prediction method does not change the training process of existing GCIL models. It is directly incorporated into them at the inference stage only, *i.e.*, our task ID predictor produces a task ID for each test sample and the existing GCIL models then perform intra-task classification in the predicted task. Without loss of generality, a parameter regularization-based method (TWP [55]) and a memory-replay method (DeLoMe [65]) are used as exemplars for this experiment. The results are shown in Table 6.2. We can see that both AA and AF performance of the two existing GCIL models are largely enhanced by the proposed task identification module. For relatively weak GCIL models like TWP, the improvement is much more substantial than the strong ones like DeLoMe. The reason is that being able to predict the task ID accurately enables the subse-

Table 6.2 : AA and AF results of enabling existing GCIL methods with our task ID prediction (TP).

Methods	CoraFull		Arxiv		Reddit		Products	
	AA/%↑	AF/%↑	AA/%↑	AF/%↑	AA/%↑	AF/%↑	AA/%↑	AF/%↑
TWP	62.6±2.2	-30.6±4.3	6.7±1.5	-50.6±13.2	8.0±5.2	-18.8±9.0	14.1±4.0	-11.4±2.0
+TP	94.3±0.9	-1.6±0.4	89.4±0.4	0.0±0.3	78.0±18.5	-0.2±0.4	81.8±3.3	-0.3±0.8
DeLoMe	81.0±0.2	-3.3±0.3	50.6±0.3	5.1±0.4	97.4±0.1	-0.1±0.1	67.5±0.7	-17.3±0.3
+TP	95.4±0.1	2.0±0.6	90.4±0.3	-1.1±0.2	99.4±0.0	-0.1±0.0	94.8±0.1	-2.2±0.2

quent CIL classification within the original task space of the test graph, not the space containing all the learned classes, significantly simplifying (reducing) the classification space. Essentially, such a task ID prediction converts the GCIL task into the GTIL task, so that much better AA and AF results are expected.

6.4.2 Ablation Study

Importance of Task ID Prediction. In GCIL, the test samples are required to be classified into one of all the learned classes. To evaluate the importance of task ID prediction that helps confine the classification space of the test samples to the classes of the predicted task, we conduct the experiments of TPP without the proposed task profiling approach and report the results in Table 6.3. Specifically, we obtain the class probabilities of the test sample for all tasks and prompts, and the class with the highest probability is treated as the class for the test sample. As shown in the table, this TPP variant can barely work on all four datasets. This is mainly due to that the graph prompts are learned task by task during training. Without the guidance of task identification, the non-normalized within-task prediction probabilities obtained with corresponding prompts pose great challenges for classifying the test samples into the correct classes.

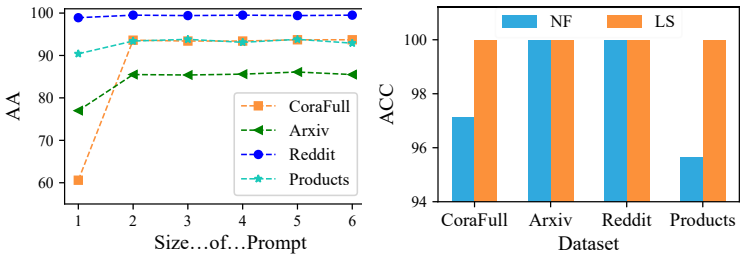
Importance of Graph Prompting. Besides the task ID prediction, we also

Table 6.3 : Results of TPP and its variants on ablating task ID prediction and graph prompting modules.

Task ID Prediction	Graph Prompting		CoraFull		Arxiv		Reddit		Products	
	Prompt	Classification Head	AA/%↑	AF/%↑	AA/%↑	AF/%↑	AA/%↑	AF/%↑	AA/%↑	AF/%↑
×	✓	✓	2.0	-5.5	3.0	-10.9	2.8	-16.8	2.7	-8.2
✓	×	×	50.7	0.0	54.0	0.0	47.4	0.0	51.8	0.0
✓	×	✓	73.8	0.0	76.3	0.0	98.6	0.0	90.0	0.0
✓	✓	×	92.8	0.0	82.9	0.0	99.0	0.0	90.7	0.0
✓	✓	✓	93.4	0.0	85.4	0.0	99.5	0.0	94.0	0.0

evaluate the importance of graph prompting. There are two modules for each task in the proposed graph promoting, *i.e.*, graph prompt Φ^t and classification head φ^t. The results with and without each module are shown in Table 6.3. We can see that the TPP variant without both Φ^t and φ^t, which is equivalent to the direct use of the GNN backbone learned only from the first task for all subsequent tasks, achieves the worst AA performance, though it is free of forgetting since there is no model updating. By incorporating either Φ^t or φ^t, the performance can be largely improved, which can be attributed to the transferable knowledge in the pre-trained GNN f and the effective adaptation of the prompts or the classifier to the subsequent tasks. Note that the variant with only Φ^t obtains much better performance than that with only φ^t, demonstrating that the learned graph prompts can more effectively model the task-specific information and bridge the gap between the first task and subsequent tasks. The results also explain the visualization of node embeddings with and without the graph prompt in Figure 6.1, where the graph prompt can largely enhance the intra-task separation. By integrating all the components, the full TPP model achieves the best performance across all datasets.

Sensitivity w.r.t the Size of Graph Prompts. We evaluate the sensitivity of the proposed method w.r.t the size of the graph prompt, *i.e.*, the number of tokens per prompt. We vary k in the range of $[1, 6]$ to verify the sensitivity and report the



(a) (b)

Figure 6.4 : **(a)** The AA results of TPP w.r.t. the size of the graph prompts. **(b)** Task ID prediction accuracy on all four datasets using Laplacian smoothing (LS) and its variant based on solely node features (NF).

results in Figure 6.4(a). It is clear that the performance of TPP increases quickly from $k = 1$ to $k = 2$ and remains stable when $k > 2$, demonstrating that TPP can be effectively adapted to different tasks with a small size of prompt for each task. This also demonstrates the transferability of the learned GNN backbone for all tasks.

Accuracy of Task ID Prediction. We further evaluate the accuracy of the proposed task ID prediction method. We compare it to a variant of our method that constructs the task prototype based on the node attributes without considering the graph structure. In this variant, each task prototype is constructed by simply averaging the attributes of the training nodes of each task. The task prototype of a test graph is constructed with the test nodes in the same way. The inference process remains the same as the proposed method. The results of these two methods are shown in Figure 6.4(b). We see that the task identification solely based on node attributes achieves high accuracy for all datasets and even predicts all of the tasks correctly for Arxiv and Reddit. This is largely attributed to the discriminative node attributes between tasks in these datasets, as demonstrated in Figure 6.3. However, it fails to discriminate tasks with similar node attributes. By contrast, the proposed

Table 6.4 : Results of average performance of TPP and Oracle Model on datasets with various task formulations.

Task Formulation	Method	CoraFull	Arxiv	Reddit	Prodcuts
Ascending Order	TPP	93.4	85.4	99.5	94.0
Ascending Order	Oracle Model	95.5	90.3	99.5	95.3
Descending Order	TPP	94.5	85.9	99.4	93.9
Descending Order	Oracle Model	96.1	91.6	99.5	94.7
Random Order	TPP	94.8	86.9	99.5	85.9
Random Order	Oracle Model	95.3	91.3	99.7	86.8

method based on Laplacian smoothing can handle all the cases, resulting in perfect task ID prediction for all tasks and datasets, which builds a strong foundation for superior GCIL performance of TPP.

Performance of TPP with different task formulations. For the task formulation, we set each task to contain two different classes of nodes and follow the commonly used task formulation strategy in [10, 65] to have fair comparisons with the baselines. Specifically, given a graph dataset with several classes, we split these classes into different tasks in numerically ascending order of the original classes, *i.e.*,classes 0 and 1 form the first task, classes 2 and 3 form the second task, and so on. To evaluate the performance of TPP with different task formulations, we further perform the class splitting in two other manners, including numerically descending and random ordering of the two classes per task. In Table 6.4, we report the average performance of the TPP and the Oracle Model with different task formulations.

From the table, we observe that the proposed TPP method can still achieve comparable performance to the Oracle Model with different task formulations, highlighting the robustness and effectiveness of TPP w.r.t. the formulation of individual tasks. Note that the performances of TPP and the Oracle Model both drop on Prod-

Table 6.5 : The accuracy of task prediction with other task formulations.

Task Formulation	CoraFull	Arxiv	Reddit	Prodcuts
Ascending Order (Reported)	100	100	100	100
Descending Order	100	100	100	100
Random Order	100	100	100	100

ucts with random task formulation. This is attributed to the heavily imbalanced class distribution of Products and the performance is evaluated by the balanced classification accuracy. Specifically, for Products, some classes contain hundreds of thousands of nodes while the number of nodes in some classes is less than 100. The ascending and descending task formulations have a relatively balanced class distribution for each task. However, the random task formulation results in some tasks with heavily imbalanced class distribution. To address this problem, debiased learning is required and we leave it for future research. Please also note that TPP learns the GNN backbone only on the first task and is frozen during the subsequent prompt learning. Different task formulations result in the GNN backbone being learned with different first tasks. The above results also reveal that the proposed graph prompting enables the learned GNN backbone to effectively adapt to all subsequent tasks despite the backbone being learned on different initial tasks.

Accuracy of task prediction with different task formulations. We further evaluate the accuracy of task prediction with different task formulations, *i.e.*, numerically descending and random ordering. The results are shown in Table 6.5. The results demonstrate that the proposed TP can accurately predict the task IDs in terms of all formulations.

Comparison to Task-Specific Models Another straightforward way to overcome forgetting is to learn task-specific models for each task. We further compare the number of additional parameters and the performance of the proposed graph

Table 6.6 : Additional parameters and performance (AA%) of the proposed graph prompting and task-specific models.

Method	Additional Parameters	CoraFull	Arxiv	Reddit	Products
Task-specific Models	$(F+d)dT$	94.3	86.8	99.5	96.3
Graph Prompting	$3FT$	93.4	85.4	99.5	94.0

Table 6.7 : Total training time and inference time (seconds) for different methods on CoraFull.

Methods	TWP	SSM	DeMoLe	TPP (Ours)
Training Time	151.6	254.9	304.6	23.6
Inference Time	0.3	0.3	0.3	0.4

prompting with task-specific models, besides the parameters of the backbone. The two methods can both achieve forget-free for GCIL with the proposed task identification. The results are reported in Table 6.6 where F is the dimensionality of the node attribute, d is the number of hidden units of SGC and T denotes the number of tasks. From the table, we can see that the proposed methods can achieve very close performance to task-specific models while introducing significantly small additional parameters for all tasks in GCIL.

Time Complexity Analysis

In Table 6.7, we report the total training time and inference time of all tasks on the CoraFull dataset, with representative models TWP [55], SSM [61] and DeLoMe [65] as the baselines, where TWP is a regularization-based method and the other two baselines are replay-based methods. From the table, we can see that replay-based methods require more time for training. This is because they typically need to construct the memory buffer based on different strategies for replaying with the new graph data. Moreover, these memory buffers accumulate to store information

from all learned tasks, resulting in the size of them becoming larger with more tasks learned. Notably, our method requires the smallest amount of training time as it does not introduce replaying memory and regularization terms. As for the inference time, the three baselines require the same amount of time as they all learn a model for all tasks. Our method requires slightly more inference time due to its task ID prediction module. Thus, there is a computational overhead in the inference of our method TPP, but it is trivial.

6.5 Conclusion

This chapter proposes a novel approach for GCIL via task profiling and prompting. The absence of task IDs during inference poses significant challenges for GCIL. To address this issue, this chapter proposes a novel Laplacian smoothing-based graph task profiling approach for GCIL, where each task is modeled by a task prototype based on Laplacian smoothing over the graph. We prove theoretically that the task prototypes of the same graph task are nearly the same with a large smoothing step and the prototypes of different tasks are distinct due to the differences in graph structure and node attributes, ensuring accurate task ID prediction for GCIL. To avoid catastrophic forgetting and achieve high within-task prediction, we further propose the first graph prompting method for GCIL which is learned to absorb the within-task information into the small task-specific graph prompts. This results in a memory-efficient TPP as i) no memory buffer is required for data replay due to its replay-free characteristic and ii) the graph prompting only requires the training of a single GNN once and a small number of tokens per prompt for each task. Extensive experiments show that TPP is fully forget-free and significantly outperforms the state-of-the-art baselines for GCIL.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This thesis focuses on deep learning on abnormal and evolving graphs. Driven by four important problems associated with existing works, four different methods are proposed in correspondence to each of them. Specifically, to enhance the performance of graph-level anomaly detection, a hierarchical memory network is designed to learn hierarchical node and graph memory modules. These memory modules explicitly capture hierarchical normal patterns of graphs by jointly minimizing graph reconstruction and graph approximation errors, enabling effective detection of both locally- and globally-anomalous graphs. To enable the generalist graph anomaly detection, a novel zero-shot generalist method is proposed, which trains one detector on a single dataset and can effectively generalize to other unseen target graphs without any further re-training or labeled nodes of target graphs during inference. To overcome the limitations of existing memory-replay approaches for continual learning, a lossless and debiased memory-replay method is designed to capture the holistic information of previous graphs and conduct replay in a debiased way. Finally, to compensate for the absence of task identifiers in class-incremental learning, a task profiling and graph prompting method is proposed to predict the task identifiers and retrieve corresponding prompts for task-specific prediction. All the methods in this thesis have achieved state-of-the-art performance, which is demonstrated through extensive experiments.

7.2 Future Work

Although the proposed generalist anomaly detection model achieves effectiveness on certain datasets, it faces three notable limitations. Firstly, the proposed UNPrompt focuses only on node-level anomaly detection, leaving other anomaly detection tasks unexplored, such as graph- and edge-level anomaly detection tasks. Secondly, the generalist model training is performed on a limited number of small-scale datasets. As a result, the learned model has limited anomaly detection capability, since its understanding of normal and abnormal patterns is constrained by the training data. Thirdly, existing methods are unable to adapt when new training datasets become available. A straightforward way to incorporate the new datasets is to relearn the model with all training graphs. However, this would induce expensive storage and computational costs. While fine-tuning the model on new datasets is another option, it may lead to the model forgetting previously learned knowledge.

The future work aims to address these issues by developing a more universal and effective graph foundation model for GAD. To enable the universal detection capacity, a unified template will be designed to reformulate multi-level detection tasks into the same format so that the universal capacity can be achieved. Moreover, inspired by the scaling law, the graph foundation model will be trained on a large collection of graphs from different domains and distributions. In this way, the foundation model would capture diverse and comprehensive knowledge of normal and abnormal graph patterns, leading to improved performance. Specifically, a mixture-of-experts architecture will be adopted to scale up the graph foundation model and capture comprehensive knowledge from the training graphs. Furthermore, to continually incorporate new training graphs into the learned model, continual learning techniques will be used to enable ongoing training of the graph foundation model, ensuring new knowledge is integrated without disrupting previously learned information.

Bibliography

[1] F. Xia, K. Sun, S. Yu, A. Aziz, L. Wan, S. Pan, and H. Liu, "Graph learning: A survey," *IEEE Transactions on Artificial Intelligence*, vol. 2, no. 2, pp. 109–127, 2021.

[2] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 1, pp. 249–270, 2020.

[3] Y. Rong, T. Xu, J. Huang, W. Huang, H. Cheng, Y. Ma, Y. Wang, T. Derr, L. Wu, and T. Ma, "Deep graph learning: Foundations, advances and applications," in *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, 2020, pp. 3555–3556.

[4] C. C. Aggarwal, H. Wang *et al.*, *Managing and mining graph data*. Springer, 2010, vol. 40.

[5] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *arXiv preprint arXiv:1709.05584*, 2017.

[6] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[7] L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: a survey," *Data mining and knowledge discovery*, vol. 29, pp. 626–688, 2015.

[8] X. Ma, J. Wu, S. Xue, J. Yang, C. Zhou, Q. Z. Sheng, H. Xiong, and L. Akoglu, "A comprehensive survey on graph anomaly detection with deep learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 12, pp. 12 012–12 038, 2021.

[9] H. Qiao, H. Tong, B. An, I. King, C. Aggarwal, and G. Pang, "Deep graph anomaly detection: A survey and new perspectives," *arXiv preprint arXiv:2409.09957*, 2024.

[10] X. Zhang, D. Song, and D. Tao, "Cglb: Benchmark tasks for continual graph learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 13 006–13 021, 2022.

[11] F. G. Febrinanto, F. Xia, K. Moore, C. Thapa, and C. Aggarwal, "Graph lifelong learning: A survey," *IEEE Computational Intelligence Magazine*, vol. 18, no. 1, pp. 32–51, 2023.

[12] Z. Tian, D. Zhang, and H.-N. Dai, "Continual learning on graphs: A survey," *arXiv preprint arXiv:2402.06330*, 2024.

[13] L. Wang, X. Zhang, H. Su, and J. Zhu, "A comprehensive survey of continual learning: Theory, method and application," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[14] G. Kim, C. Xiao, T. Konishi, Z. Ke, and B. Liu, "A theoretical study on solving continual learning," *Advances in neural information processing systems*, vol. 35, pp. 5065–5079, 2022.

[15] G. Kim, C. Xiao, T. Konishi, and B. Liu, "Learnability and algorithm for continual learning," in *International Conference on Machine Learning*. PMLR, 2023, pp. 16 877–16 896.

[16] H. Lin, Y. Shao, W. Qian, N. Pan, Y. Guo, and B. Liu, "Class incremental learning via likelihood ratio based task prediction," in *The Twelfth International Conference on Learning Representations*, 2024.

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

[18] C. C. Aggarwal and H. Wang, "Graph data management and mining: a survey of algorithms and applications," in *Managing and Mining Graph Data*. Springer, 2010, pp. 13–68.

[19] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE signal processing magazine*, vol. 30, no. 3, pp. 83–98, 2013.

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

[21] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[22] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[23] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[24] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: Higher-order graph neural networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 4602–4609.

[25] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman, "Provably powerful graph networks," *Advances in neural information processing systems*, vol. 32, 2019.

[26] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[27] K. Liu, Y. Dou, Y. Zhao, X. Ding, X. Hu, R. Zhang, K. Ding, C. Chen, H. Peng, K. Shu *et al.*, "Bond: Benchmarking unsupervised outlier node detection on static attributed graphs," *Advances in Neural Information Processing Systems*, vol. 35, pp. 27 021–27 035, 2022.

[28] K. Ding, J. Li, R. Bhanushali, and H. Liu, "Deep anomaly detection on attributed networks," in *Proceedings of the 2019 SIAM international conference on data mining*. SIAM, 2019, pp. 594–602.

[29] H. Fan, F. Zhang, and Z. Li, "Anomalydae: Dual autoencoder for anomaly detection on attributed networks," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 5685–5689.

[30] Y. Liu, Z. Li, S. Pan, C. Gong, C. Zhou, and G. Karypis, "Anomaly detection on attributed networks via contrastive self-supervised learning," *IEEE transactions on neural networks and learning systems*, vol. 33, no. 6, pp. 2378–2392, 2021.

[31] Y. Zheng, M. Jin, Y. Liu, L. Chi, K. T. Phan, and Y.-P. P. Chen, "Generative and contrastive self-supervised learning for graph anomaly detection," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 12, pp. 12 220–12 233, 2021.

[32] H. Qiao and G. Pang, "Truncated affinity maximization: One-class homophily modeling for graph anomaly detection," *Advances in Neural Information Processing Systems*, vol. 36, 2023.

[33] Y. Liu, X. Ao, Z. Qin, J. Chi, J. Feng, H. Yang, and Q. He, "Pick and choose: a gnn-based imbalanced learning approach for fraud detection," in *Proceedings of the web conference 2021*, 2021, pp. 3168–3177.

[34] Z. Chai, S. You, Y. Yang, S. Pu, J. Xu, H. Cai, and W. Jiang, "Can abnormality be detected by graph neural networks?" in *IJCAI*, 2022, pp. 1945–1951.

[35] J. Tang, J. Li, Z. Gao, and J. Li, "Rethinking graph neural networks for anomaly detection," in *International Conference on Machine Learning*. PMLR, 2022, pp. 21 076–21 089.

[36] Y. Gao, X. Wang, X. He, Z. Liu, H. Feng, and Y. Zhang, "Addressing heterophily in graph anomaly detection: A perspective of graph spectrum," in *Proceedings of the ACM Web Conference 2023*, 2023, pp. 1528–1538.

[37] M. Neumann, R. Garnett, C. Bauckhage, and K. Kersting, "Propagation kernels: efficient graph kernels from propagated information," *Machine Learning*, vol. 102, pp. 209–245, 2016.

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

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

[40] G. Pang, K. M. Ting, and D. Albrecht, "Lesinn: Detecting anomalies by identifying least similar nearest neighbours," in *2015 IEEE international conference on data mining workshop (ICDMW)*. IEEE, 2015, pp. 623–630.

[41] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 eighth ieee international conference on data mining*. IEEE, 2008, pp. 413–422.

[42] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, 2000, pp. 93–104.

[43] B. Schölkopf, R. C. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt, "Support vector method for novelty detection," *Advances in neural information processing systems*, vol. 12, 1999.

[44] L. Zhao and L. Akoglu, "On using classification datasets to evaluate graph outlier detection: Peculiar observations and new insights," *Big Data*, 2021.

[45] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft, "Deep one-class classification," in *International conference on machine learning*. PMLR, 2018, pp. 4393–4402.

[46] R. Ma, G. Pang, L. Chen, and A. van den Hengel, "Deep graph-level anomaly detection by glocal knowledge distillation," in *WSDM '22: The Fifteenth ACM International Conference on Web Search and Data Mining*, 2022.

[47] X. Luo, J. Wu, J. Yang, S. Xue, H. Peng, C. Zhou, H. Chen, Z. Li, and Q. Z.

Sheng, "Deep graph level anomaly detection with contrastive learning," *Scientific Reports*, vol. 12, no. 1, p. 19867, 2022.

[48] Y. Liu, K. Ding, H. Liu, and S. Pan, "Good-d: On unsupervised graph out-of-distribution detection," in *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining*, 2023, pp. 339–347.

[49] C. Qiu, M. Kloft, S. Mandt, and M. Rudolph, "Raising the bar in graph-level anomaly detection," *arXiv preprint arXiv:2205.13845*, 2022.

[50] Q. Zhou, G. Pang, Y. Tian, S. He, and J. Chen, "Anomalyclip: Object-agnostic prompt learning for zero-shot anomaly detection," in *The Twelfth International Conference on Learning Representations*, 2024.

[51] J. Zhu and G. Pang, "Toward generalist anomaly detection via in-context residual learning with few-shot sample prompts," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 17 826–17 836.

[52] Y. Liu, S. Li, Y. Zheng, Q. Chen, C. Zhang, and S. Pan, "Arc: A generalist graph anomaly detector with in-context learning," *arXiv preprint arXiv:2405.16771*, 2024.

[53] B. He, X. He, Y. Zhang, R. Tang, and C. Ma, "Dynamically expandable graph convolution for streaming recommendation," in *Proceedings of the ACM Web Conference 2023*, 2023, pp. 1457–1467.

[54] F. Zhou and C. Cao, "Overcoming catastrophic forgetting in graph neural networks with experience replay," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 4714–4722.

[55] H. Liu, Y. Yang, and X. Wang, "Overcoming catastrophic forgetting in graph neural networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, 2021, pp. 8653–8661.

[56] L. Sun, J. Ye, H. Peng, F. Wang, and S. Y. Philip, "Self-supervised continual graph learning in adaptive riemannian spaces," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, 2023, pp. 4633–4642.

[57] C. Wang, Y. Qiu, D. Gao, and S. Scherer, "Lifelong graph learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 13 719–13 728.

[58] J. Su, D. Zou, Z. Zhang, and C. Wu, "Towards robust graph incremental learning on evolving graphs," in *Proceedings of the 40th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 202. PMLR, 23–29 Jul 2023, pp. 32 728–32 748.

[59] A. Rakaraddi, L. Siew Kei, M. Pratama, and M. De Carvalho, "Reinforced continual learning for graphs," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, 2022, pp. 1666–1674.

[60] M. Perini, G. Ramponi, P. Carbone, and V. Kalavri, "Learning on streaming graphs with experience replay," in *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, 2022, pp. 470–478.

[61] X. Zhang, D. Song, and D. Tao, "Sparsified subgraph memory for continual graph representation learning," in *ICDM*, 2022.

[62] P. Zhang, Y. Yan, C. Li, S. Wang, X. Xie, G. Song, and S. Kim, "Continual learning on dynamic graphs via parameter isolation," in *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in*

Information Retrieval, ser. SIGIR '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 601–611.

[63] X. Zhang, D. Song, and D. Tao, "Ricci curvature-based graph sparsification for continual graph representation learning," *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[64] Y. Liu, R. Qiu, and Z. Huang, "Cat: Balanced continual graph learning with graph condensation," *arXiv preprint arXiv:2309.09455*, 2023.

[65] C. Niu, G. Pang, and L. Chen, "Graph continual learning with debiased lossless memory replay," *arXiv preprint arXiv:2404.10984*, 2024.

[66] D. Abati, J. Tomczak, T. Blankevoort, S. Calderara, R. Cucchiara, and B. E. Bejnordi, "Conditional channel gated networks for task-aware continual learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 3931–3940.

[67] J. Von Oswald, C. Henning, B. F. Grewe, and J. Sacramento, "Continual learning with hypernetworks," *arXiv preprint arXiv:1906.00695*, 2019.

[68] C. Henning, M. Cervera, F. D'Angelo, J. Von Oswald, R. Traber, B. Ehret, S. Kobayashi, B. F. Grewe, and J. Sacramento, "Posterior meta-replay for continual learning," *Advances in neural information processing systems*, vol. 34, pp. 14 135–14 149, 2021.

[69] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," *Advances in neural information processing systems*, vol. 31, 2018.

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

[71] K. Ding, J. Li, N. Agarwal, and H. Liu, "Inductive anomaly detection on attributed networks," in *Proceedings of the twenty-ninth international conference on international joint conferences on artificial intelligence*, 2021, pp. 1288–1294.

[72] A. Kumagai, T. Iwata, and Y. Fujiwara, "Semi-supervised anomaly detection on attributed graphs," in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–8.

[73] M. Jin, Y. Liu, Y. Zheng, L. Chi, Y.-F. Li, and S. Pan, "Anemone: Graph anomaly detection with multi-scale contrastive learning," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021, pp. 3122–3126.

[74] Q. Wang, G. Pang, M. Salehi, W. Buntine, and C. Leckie, "Cross-domain graph anomaly detection via anomaly-aware contrastive alignment," *arXiv preprint arXiv:2212.01096*, 2022.

[75] H. Qiao and G. Pang, "Truncated affinity maximization: One-class homophily modeling for graph anomaly detection," *arXiv preprint arXiv:2306.00006*, 2023.

[76] C. Y. Lee and Y.-P. P. Chen, "Descriptive prediction of drug side-effects using a hybrid deep learning model," *International Journal of Intelligent Systems*, vol. 36, no. 6, pp. 2491–2510, 2021.

[77] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[78] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," *Advances in neural information processing systems*, vol. 19, 2006.

[79] S. Zhai, Y. Cheng, W. Lu, and Z. Zhang, "Deep structured energy based models for anomaly detection," in *International conference on machine learning*. PMLR, 2016, pp. 1100–1109.

[80] J. Chen, S. Sathe, C. Aggarwal, and D. Turaga, "Outlier detection with autoencoder ensembles," in *Proceedings of the 2017 SIAM international conference on data mining*. SIAM, 2017, pp. 90–98.

[81] C. Zhou and R. C. Paffenroth, "Anomaly detection with robust deep autoencoders," in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017, pp. 665–674.

[82] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu, D. Cho, and H. Chen, "Deep autoencoding gaussian mixture model for unsupervised anomaly detection," in *International conference on learning representations*, 2018.

[83] D. Gong, L. Liu, V. Le, B. Saha, M. R. Mansour, S. Venkatesh, and A. v. d. Hengel, "Memorizing normality to detect anomaly: Memory-augmented deep autoencoder for unsupervised anomaly detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1705–1714.

[84] H. Park, J. Noh, and B. Ham, "Learning memory-guided normality for anomaly detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 14 372–14 381.

[85] J. Weston, S. Chopra, and A. Bordes, "Memory networks," in *International Conference on Learning Representations*, 2015.

[86] C. Li, J. Zhu, and B. Zhang, "Learning to generate with memory," in *International conference on machine learning*. PMLR, 2016, pp. 1177–1186.

[87] Y. Kim, M. Kim, and G. Kim, "Memorization precedes generation: Learning unsupervised GANs with memory networks," in *International Conference on Learning Representations*, 2018.

[88] Z. Wu, Y. Xiong, S. X. Yu, and D. Lin, "Unsupervised feature learning via non-parametric instance discrimination," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 3733–3742.

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

[90] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*, 2018.

[91] G. Pang, C. Shen, and A. van den Hengel, "Deep anomaly detection with deviation networks," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 353–362.

[92] G. O. Campos, A. Zimek, J. Sander, R. J. Campello, B. Micenková, E. Schubert, I. Assent, and M. E. Houle, "On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study," *Data mining and knowledge discovery*, vol. 30, pp. 891–927, 2016.

[93] R. F. Woolson, "Wilcoxon signed-rank test," *Wiley encyclopedia of clinical trials*, pp. 1–3, 2007.

[94] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel, "Deep learning for anomaly detection: A review," *ACM computing surveys (CSUR)*, vol. 54, no. 2, pp. 1–38, 2021.

[95] J. J. McAuley and J. Leskovec, "From amateurs to connoisseurs: modeling

the evolution of user expertise through online reviews," in *Proceedings of the 22nd international conference on World Wide Web*, 2013, pp. 897–908.

[96] S. Rayana and L. Akoglu, "Collective opinion spam detection: Bridging review networks and metadata," in *Proceedings of the 21th acm sigkdd international conference on knowledge discovery and data mining*, 2015, pp. 985–994.

[97] Y. Yang, Y. Xu, Y. Sun, Y. Dong, F. Wu, and Y. Zhuang, "Mining fraudsters and fraudulent strategies in large-scale mobile social networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, no. 1, pp. 169–179, 2019.

[98] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations*, 2016.

[99] Z. Peng, M. Luo, J. Li, H. Liu, Q. Zheng *et al.*, "Anomalous: A joint modeling approach for anomaly detection on attributed networks." in *IJCAI*, vol. 18, 2018, pp. 3513–3519.

[100] T. Huang, Y. Pei, V. Menkovski, and M. Pechenizkiy, "Hop-count based self-supervised anomaly detection on attributed networks," in *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 2022, pp. 225–241.

[101] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, "Learning transferable visual models from natural language supervision," in *International conference on machine learning*. PMLR, 2021, pp. 8748–8763.

[102] J. Liu, C. Yang, Z. Lu, J. Chen, Y. Li, M. Zhang, T. Bai, Y. Fang, L. Sun, P. S. Yu *et al.*, "Towards graph foundation models: A survey and beyond," *arXiv preprint arXiv:2310.11829*, 2023.

[103] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–35, 2023.

[104] X. Sun, H. Cheng, J. Li, B. Liu, and J. Guan, "All in one: Multi-task prompting for graph neural networks," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023, pp. 2120–2131.

[105] Z. Liu, X. Yu, Y. Fang, and X. Zhang, "Graphprompt: Unifying pre-training and downstream tasks for graph neural networks," in *Proceedings of the ACM Web Conference 2023*, 2023, pp. 417–428.

[106] Y. Li, P. Wang, Z. Li, J. X. Yu, and J. Li, "Zerog: Investigating cross-dataset zero-shot transferability in graphs," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2024, pp. 1725–1735.

[107] H. Zhao, A. Chen, X. Sun, H. Cheng, and J. Li, "All in one and one for all: A simple yet effective method towards cross-domain graph pretraining," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2024, pp. 4443–4454.

[108] T. Fang, Y. Zhang, Y. Yang, C. Wang, and L. Chen, "Universal prompt tuning for graph neural networks," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[109] G. W. Stewart, "On the early history of the singular value decomposition," *SIAM review*, vol. 35, no. 4, pp. 551–566, 1993.

[110] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley interdisciplinary reviews: computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[111] A. Li, C. Qiu, M. Kloft, P. Smyth, M. Rudolph, and S. Mandt, "Zero-shot anomaly detection via batch normalization," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[112] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *International conference on machine learning*. PMLR, 2019, pp. 6861–6871.

[113] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang, "Deep graph contrastive representation learning," *arXiv preprint arXiv:2006.04131*, 2020.

[114] Z. Xu, X. Huang, Y. Zhao, Y. Dong, and J. Li, "Contrastive attributed network anomaly detection with data augmentation," in *Pacific-Asia conference on knowledge discovery and data mining*. Springer, 2022, pp. 444–457.

[115] S. Kumar, X. Zhang, and J. Leskovec, "Predicting dynamic embedding trajectory in temporal interaction networks," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 1269–1278.

[116] P. I. Sánchez, E. Müller, F. Laforet, F. Keller, and K. Böhm, "Statistical selection of congruent subspaces for mining attributed graphs," in *2013 IEEE 13th international conference on data mining*. IEEE, 2013, pp. 647–656.

[117] J. Chen, G. Zhu, C. Yuan, and Y. Huang, "Boosting graph anomaly detection with adaptive message passing," in *The Twelfth International Conference on Learning Representations*, 2024.

[118] J. Tang, F. Hua, Z. Gao, P. Zhao, and J. Li, "Gadbench: Revisiting and benchmarking supervised graph anomaly detection," *Advances in Neural Information Processing Systems*, vol. 36, pp. 29 628–29 653, 2023.

[119] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation-based anomaly detection," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 6, no. 1, pp. 1–39, 2012.

[120] X. Luo, J. Wu, A. Beheshti, J. Yang, X. Zhang, Y. Wang, and S. Xue, "Comga: Community-aware attributed graph anomaly detection," in *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, 2022, pp. 657–665.

[121] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI open*, vol. 1, pp. 57–81, 2020.

[122] M. Farajtabar, N. Azizan, A. Mott, and A. Li, "Orthogonal gradient descent for continual learning," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020, pp. 3762–3773.

[123] S. Yan, J. Xie, and X. He, "Der: Dynamically expandable representation for class incremental learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 3014–3023.

[124] O. Ostapenko, M. Puscas, T. Klein, P. Jahnichen, and M. Nabi, "Learning to remember: A synaptic plasticity driven framework for continual learning," in

Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2019, pp. 11 321–11 329.

[125] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[126] Z. Ke and B. Liu, "Continual learning of natural language processing tasks: A survey," *arXiv preprint arXiv:2211.12701*, 2022.

[127] X. Zhang, D. Song, and D. Tao, "Continual learning on graphs: Challenges, solutions, and opportunities," *arXiv preprint arXiv:2402.11565*, 2024.

[128] W. Jin, X. Tang, H. Jiang, Z. Li, D. Zhang, J. Tang, and B. Yin, "Condensing graphs via one-step gradient matching," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 720–730.

[129] W. Jin, L. Zhao, S. Zhang, Y. Liu, J. Tang, and N. Shah, "Graph condensation for graph neural networks," in *International Conference on Learning Representations*, 2022.

[130] X. Gao, T. Chen, Y. Zang, W. Zhang, Q. V. H. Nguyen, K. Zheng, and H. Yin, "Graph condensation for inductive node representation learning," *arXiv preprint arXiv:2307.15967*, 2023.

[131] M. Liu, S. Li, X. Chen, and L. Song, "Graph condensation via receptive field distribution matching," *arXiv preprint arXiv:2206.13697*, 2022.

[132] X. Zheng, M. Zhang, C. Chen, Q. V. H. Nguyen, X. Zhu, and S. Pan, "Structure-free graph condensation: From large-scale graphs to condensed

graph-free data," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[133] A. K. Menon, S. Jayasumana, A. S. Rawat, H. Jain, A. Veit, and S. Kumar, "Long-tail learning via logit adjustment," in *International Conference on Learning Representations*, 2021.

[134] L. Galke, I. Vagliano, B. Franke, T. Zielke, M. Hoffmann, and A. Scherp, "Lifelong learning on evolving graphs under the constraints of imbalanced classes and new classes," *Neural Networks*, vol. 164, pp. 156–176, 2023.

[135] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore, "Automating the construction of internet portals with machine learning," *Information Retrieval*, vol. 3, pp. 127–163, 2000.

[136] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," *Advances in neural information processing systems*, vol. 33, pp. 22 118–22 133, 2020.

[137] A. Sinha, Z. Shen, Y. Song, H. Ma, D. Eide, B.-J. Hsu, and K. Wang, "An overview of microsoft academic service (mas) and applications," in *Proceedings of the 24th international conference on world wide web*, 2015, pp. 243–246.

[138] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 12, pp. 2935–2947, 2017.

[139] D. Lopez-Paz and M. Ranzato, "Gradient episodic memory for continual learning," *Advances in neural information processing systems*, vol. 30, 2017.

[140] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars, "Memory aware synapses: Learning what (not) to forget," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 139–154.

[141] X. Zhang, D. Song, and D. Tao, "Hierarchical prototype networks for continual graph representation learning," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 45, no. 04, pp. 4622–4636, 2023.

[142] X. Sun, J. Zhang, X. Wu, H. Cheng, Y. Xiong, and J. Li, "Graph prompt learning: A comprehensive survey and beyond," *arXiv preprint arXiv:2311.16534*, 2023.

[143] T. Fang, Y. Zhang, Y. Yang, C. Wang, and L. Chen, "Universal prompt tuning for graph neural networks," in *Advances in Neural Information Processing Systems*, vol. 36, 2023, pp. 52 464–52 489.

[144] Z. Wang, Z. Zhang, C.-Y. Lee, H. Zhang, R. Sun, X. Ren, G. Su, V. Perot, J. Dy, and T. Pfister, "Learning to prompt for continual learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 139–149.

[145] Z. Wang, Z. Zhang, S. Ebrahimi, R. Sun, H. Zhang, C.-Y. Lee, X. Ren, G. Su, V. Perot, J. Dy *et al.*, "Dualprompt: Complementary prompting for rehearsal-free continual learning," in *European Conference on Computer Vision*. Springer, 2022, pp. 631–648.

[146] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, "Graph contrastive learning with augmentations," *Advances in neural information processing systems*, vol. 33, pp. 5812–5823, 2020.

[147] F. R. Chung, *Spectral graph theory*. American Mathematical Soc., 1997, vol. 92.

[148] U. Von Luxburg, "A tutorial on spectral clustering," *Statistics and computing*, vol. 17, pp. 395–416, 2007.