Improving representation learning on graph-structural data for classification, generation, and recommendation

Luo, Tianze

2024

Luo, T. (2024). Improving representation learning on graph-structural data for classification, generation, and recommendation. Doctoral thesis, Nanyang Technological University, Singapore. https://hdl.handle.net/10356/179453

https://hdl.handle.net/10356/179453

https://doi.org/10.32657/10356/179453

Improving Representation Learning on Graph-Structural Data for Classification, Generation, and Recommendation

Tianze Luo

School of Computer Science and Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Doctor of Philosophy (Ph.D)

2024

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

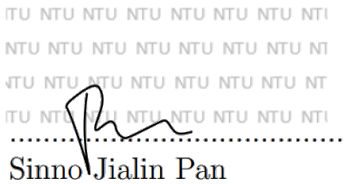
18 January 2024
.....
Date

.....
Tianze Luo

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

24 Jun 2024
.....
Date



.....
Sinno Jialin Pan

Authorship Attribution Statement

This thesis contains material from four papers published or under review in the following peer-reviewed conferences/journals in which I am listed as the first author.

Chapter 3 is under review as Tianze Luo, Qiuhao Zeng, Tianbo Li, Sinno Jialin Pan."Meta-Contrast for Graph Representation Learning". Submitted to IEEE Transactions on Pattern Analysis and Machine Intelligence (IEEE TPAMI).

The contributions of the co-authors are as follows:

- I proposed the problem and the model.
- Qiuhao Zeng and I implemented the model and ran the experiments.
- I wrote the manuscript.
- Qiuhao Zeng, Dr. Tianbo Li, and Prof Sinno Jialin Pan provided insightful comments and revised the manuscript.

Chapter 4 is published as Tianze Luo, Zhanfeng Mo, Sinno Jialin Pan. "Learning Adaptive Multiresolution Transforms via Meta-Framelet-based Graph Convolutional Network". Accepted by the International Conference on Learning Representations (ICLR) (2024).

The contributions of the co-authors are as follows:

- I proposed the problem and the model.
- Zhanfeng Mo revised the algorithm and justified the model.
- I implemented the model and ran the experiments.
- Zhanfeng Mo and I wrote the manuscript.
- Prof Sinno Jialin Pan provided insightful comments and revised the manuscript.

Chapter 5 is published as Tianze Luo, Zhanfeng Mo, Sinno Jialin Pan. "Fast Graph Generation via Spectral Diffusion". Accepted by the IEEE Transactions on Pattern Analysis and Machine Intelligence (IEEE TPAMI) (2023).

The contributions of the co-authors are as follows:

- I proposed the problem and the model.
- Zhanfeng Mo justified the model and provided the proof.
- I implemented the model and ran the experiments.
- Zhanfeng Mo and I wrote the manuscript.
- Prof Sinno Jialin Pan provided insightful comments and revised the manuscript.

Chapter 6 is published as Tianze Luo, Yong Liu, Sinno Jialin Pan. "Collaborative Sequential Recommendations via Multi-view GNN-Transformers". Accepted by ACM Transactions on Information Systems (ACM TOIS).

The contributions of the co-authors are as follows:

- I proposed the problem and the model.
- I implemented the model and ran the experiments.
- I wrote the manuscript.
- Prof Sinno Jialin Pan and Dr. Yong Liu provided insightful comments and revised the manuscript.

23-Jan-2024

Date

Tianze Luo

Abstract

This thesis explores innovative approaches in graph representation learning and its applications using deep learning models, making significant contributions across several key areas. We first introduce the Graph Meta-Contrast (GMeCo) framework, a novel meta-learning framework for contrastive representation learning on graphs. GMeCo effectively generates augmented graphs and maximizes the mutual information between the augmented graphs and input graphs, outperforming current methods in robust and discriminative feature learning.

Next, we present the Multiresolution Meta-Framelet-based Graph Convolutional Network (MM-FGCN) model. This model represents an advancement in adaptive multiresolution analysis of graphs, overcoming fixed transform limitations and dynamically handling graph data at various scales. MM-FGCN's ability to capture both micro- and macro-level graph structures shows its superiority in various graph learning tasks.

Furthermore, we introduce Graph Spectral Diffusion Model (GSDM), a novel approach for graph-structured data generation. GSDM utilizes low-rank diffusion Stochastic Differential Equations in graph spectrum space, enhancing graph topology generation and reducing computational load. This method demonstrates improved efficiency and quality in graph generation compared to existing models.

Lastly, we develop a novel framework for sequential recommendation systems through a multi-view approach, combining Graph Neural Networks (GNNs) and Transformers. This multi-view structure leverages user-item interactions and collaborative information, offering robust and accurate user preference predictions. This model demonstrates its effectiveness over traditional models.

Overall, this thesis presents effective methods and models in graph representation learning, contributing to advancements in the field and laying a foundation for future research in graph-based deep learning applications.

Acknowledgments

Upon completing my Ph.D. thesis, gratitude overwhelms me. I would like to extend my heartfelt appreciation to everyone who has offered invaluable assistance throughout this journey.

I would like to express my deepest gratitude to Prof. Sinno Jialin Pan, my supervisor, for his invaluable guidance, patience, and expertise throughout my Ph.D. period. His insightful comments and critical thinking consistently inspire me when approaching research problems. His dedication to academic research serves as an exemplary model of what a true researcher should embody. All the expertise, competencies, and research acumen that I have gained from my supervisor over the years will be invaluable throughout my entire life.

I am also immensely grateful to my girlfriend Jiaxin Lu, for her unwavering support and understanding throughout the challenging journey of completing my Ph.D. degree. Her patience, encouragement, and love have been a constant source of strength and motivation. Without her by my side, this achievement would not have been possible. I am truly fortunate to have her as a pillar of support in my life.

I would like to take this chance to thank my friends and teammates for their kind friendship and support: Zhanfeng Mo, Qiuhao Zeng, Tianbo Li, Bosheng Ding, Yue Deng, Quanyu Long, Yu Chen, Jianda Chen, Xinyi Huang, Fangkai Jiao, Chengwei Qin, Wenya Wang, Haiyan Yin, Shangyu Chen, Longkai Huang, Jiangjun Zhao. Their companionship has enriched my journey at NTU, creating cherished memories and ensuring a fulfilling experience that will remain etched in my heart.

Most importantly, I want to extend my deepest gratitude to my beloved parents. Without their unwavering support and trust, I would not have been able to navigate the numerous challenges in my life. I aspire to continue striving in a way that makes them proud.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Graphs are ubiquitous data structures that exist in various aspects of our world. Various data such as protein structure, linguistics, recommendation systems, social networks, etc, can be expressed and organized by graph structures. Essentially, a graph is a combination of objects (i.e. vertices) along with a set of interactions (i.e., edges) between pairs of these objects. In the biology field, for protein structure learning, the vertices of the node are the molecules and the edges are the chemical bonds that connect each molecule pair. Similarly, in recommender systems, a user-item graph is commonly used where the vertices represent each user and item while the edges reflect user actions like purchases or clicks. Graph representations allow data to be stored and retrieved efficiently while maintaining the correlations between each object [17]. Utilizing these structures, we can analyze and comprehend the intricate connections and characteristics of each element within the network.

Graph representation learning aims to provide learning mechanisms for data with graph structures, to convert raw graph data into high dimensional vectors while preserving intrinsic graph properties [17]. With learned graph representations, one can adopt them on various downstream tasks such as applications on community detection, recommendation, node clustering, node classification, edge and link prediction, clustering, and graph prediction.

We denote a graph as $G = (V, E)$, where $V = \{v_1, ..., v_n\}$ denotes the vertices (nodes) of the graph, and $E = \{e_{i,j}\}$ denotes the set of edges. An edge $e_{i,j}$ connects vertex v_i and

v_j. The architecture of a graph is usually represented as an adjacency matrix A, which contains non-negative weights associated with each edge, i.e. $a_{i,j} \geq 0$ for $i, j \in \{1, ..., n\}$. $a_{i,j} = 0$ if no edge directly connects between the vertices v_i and v_j. Furthermore, for undirected graph, $a_{i,j} = a_{j,i}$.

Graph representation learning aims to learn a mapping function f that maps each node or entire (sub)graph to a low-dimensional vector space $\mathbb{R}^d$ where the new representation X encodes the structural information about the graph. The goal is to optimize this mapping so that geometric relationships in the embedding space reflect the structure of the original graph [67]. In the mathematical expression, for graph $G(V, E)$ we would like to find a mapping:

$$f : E \times V \rightarrow X \in \mathbb{R}^{n \times d},$$

where $d << |V|$, and $X = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$ is the learned embedding that captures the structural properties of each node.

1.1.1 Classical Graph Representation Learning

Prior to deep neural network models for graph representation learning, various classical models were designed to capture graph structures in a low dimensional space.

Key classical methods in graph representation learning include:

- Matrix Factorization Techniques: These include methods like Singular Value Decomposition (SVD) applied to adjacency or Laplacian matrices of graphs. They are foundational in understanding graph structure in a reduced dimensional space. Representative methods include Laplacian eigenmap [7], large-scale graph factorization [2], GraRep [14], and HOPE [157].

- Random Walk Approaches: Techniques like DeepWalk [162] and node2vec [61] use random walks to sample the structure of the graph. They generate sequences of nodes, akin to sentences in natural language, which are then used to learn embeddings using language modeling techniques like Word2Vec [147].

- Graph Kernels: Kernel methods, commonly used in machine learning, are employed to measure the similarity between pairs of data points. In a similar vein, graph kernels are utilized to assess the similarities between two graphs. In graph kernel methods, a feature mapping function denoted as ϕ is introduced, which implicitly maps the graphs $\mathcal{G}$ to a Hilbert space $\mathcal{H}_k$ associated with a kernel function $k(\cdot, \cdot)$. To compute the similarity between two graphs, G_1 and G_2, within the Hilbert space $\mathcal{H}_k$, one can directly calculate $k(G_1, G_2)$ without the need to transform G_1 and G_2 into the feature space $\phi(G_1)$ and $\phi(G_2)$, since $k(G_1, G_2) = \langle \phi(G_1), \phi(G_2) \rangle_{\mathcal{H}_k}$. Chapter two will provide detailed explanations of well-known graph kernels, such as the Weisfeiler-Lehman kernel, Graphlet kernel, Shortest path kernel, and Random walk kernel, among others.

- Neighborhood Aggregation: Although on the cusp of classical and modern techniques, methods like GraphSAGE [67] are worth mentioning. They aggregate feature information from a node's neighborhood to learn a representation for each node.

Classical graph representation learning methods are often unsupervised, focusing on preserving graph structure and features without the need for labeled data. However, they may struggle with scalability and handling dynamic graphs. The advent of Graph Neural Networks (GNNs) has addressed some of these limitations, but understanding classical methods provides essential foundations for appreciating more advanced techniques in graph representation learning.

1.1.2 Graph Neural Networks

Deep learning architectures designed for learning from graph structural data are commonly referred to as Graph Neural Networks (GNNs). Over recent decades, with the advancement of deep learning methods, GNN architectures have been extensively and deeply explored. Broadly, based on their architectural characteristics, research on Graph Neural Networks can be categorized into four main types: (1) Recurrent Graph Neural Networks (RecGNNs), (2) Convolutional Graph Neural Networks (ConvGNNs), (3) graph autoencoders (GAEs), and (4) spatial-temporal Graph Neural Networks (STGNNs) [228].

Recurrent Graph Neural Networks (RecGNNs) are among the pioneering works in the field of Graph Neural Networks. They focus on learning vertex representations using recurrent neural architectures, such as RNNs. These models operate under the assumption that a node within a graph continuously exchanges information or messages with its neighbors until a stable equilibrium is reached. Consequently, RecGNNs often incorporate information diffusion mechanisms, like RNNs, to iteratively update the states of nodes by exchanging information with their neighbors until the hidden state of each node converges to a stable equilibrium.

Convolutional Graph Neural Networks (ConvGNNs) employ a cyclic mutual dependency architecture with a fixed number of layers, where each layer has unique weights for aggregating neighboring nodes' features to perform message passing and information aggregation. Prominent models in this category include Graph Convolutional Networks (GCN) [100], GraphSage [67], and Graph Attention Networks (GAT) [208]. These models have been widely utilized for graph learning tasks.

Unsupervised and semi-supervised graph representation learning has made significant advancements in recent years. These methods often rely on the homophily assumption in graph structures, which states that connected nodes in a graph tend to share similar properties. This phenomenon is frequently observed in various types of graph structural data, such as social networks and recommender systems. For example, in social networks, users with many common friends are more likely to know each other, and in recommender systems, users who purchase many of the same items are likely to have similar preferences. Consequently, in graph representation learning, closely connected nodes are more likely to have the same labels, while nodes that are distant from each other tend to have different labels.

Building upon this assumption, recent developments in contrastive learning and pretraining techniques aim to distinguish between similar and dissimilar nodes based on the graph topology. Methods in this category learn node and subgraph representations with the goal of pulling together similar nodes or subgraphs while pushing apart dissimilar ones. Some notable approaches include: Deep Graph Infomax (DGI) [211], which extends Deep Infomax to graphs by learning node representations through contrasting the input graph with negative samples generated using a corruption function. InfoGraph [191], an

extension of Deep Infomax, focuses on maximizing mutual information between graph-level representations and substructures of different scales. MVGRL [71], which combines graph diffusion networks with contrastive learning by contrasting representations from the graph data with diffusion graphs. GraphCL [240], which introduces four methods for constructing positive augmentations, including node dropping, edge perturbation, attribute masking, and subgraph operations.

1.1.3 Graph Generation Models

The graph generation models aim to generate graphs that have the desired properties. The generation models can be applied to many specific domains such as social network analysis, molecule and drug discovery, traffic network analysis, protein design, program synthesis, etc.

Learning to generate graph-structural data not only requires knowing the nodes' feature distribution but also a deep understanding of the underlying graph topology, which is essential to modeling various graph instances, such as social networks [136, 137, 215, 236], molecule structures [149, 183, 245], neural architectures [113], recommender systems [126, 135], etc. Conventional likelihood-based graph generative models, e.g. GraphGAN [216], GraphVAE [186] and GraphRNN [238], have demonstrated great strength on graph generation tasks. In general, a likelihood-based model is designed to learn the likelihood function of the underlying graph data distribution, with which one can draw new samples with preserved graph properties from the distribution of interest. However, most likelihood-based generative models suffer from either limited quality of modeling graph structures, or considerable computational burden [91].

Recently, a series of diffusion-based generative models have been proposed to overcome the limitations of likelihood-based models. Although being originally established for image generation [187], diffusion models exhibit great success in graph generation tasks with complex graph structural properties [91, 153]. Roughly speaking, diffusion is a mathematical technique that involves a Stochastic Differential Equation (SDE) used to smoothly transform real data into pure noise by adding more noise. Diffusion models are a class of probabilistic generative models that use this technique in reverse to generate representative data samples from noise. These models are trained to learn the underlying

structure of a dataset by simulating how data points gradually become blurry due to added noise. Therefore, a well-trained diffusion model allows us to restore the original data by reversing the diffusion process and gradually removing the noise from the blurred sample.

The first graph diffusion model through SDEs, coined Graph Diffusion via SDE Systems (GDSS) [91], is designed to simultaneously generate node features and adjacency matrix via reversed diffusion. Similar to image diffusion models [187, 188], at each diffusion step, GDSS directly inserts standard Gaussian noise to both node features and the adjacency matrix. Meanwhile, two separate neural networks are trained to learn the score functions of the node features and adjacency matrix, respectively.

However, unlike the densely distributed image data, graph adjacency matrices can be highly sparse, which makes isotropic Gaussian noise insertion incompatible with graph structural data. In these circumstances, there is a stark difference between the diffusion process on images and on graph adjacency matrices. As can be seen from the figure, the image corrupted by full-rank Gaussian noise exhibits recognizable numerical patterns along the early- and middle-stage of forward diffusion. However, the corrupted sparse graph adjacency matrix degenerates into a dense matrix with uniformly distributed entries in a few diffusion steps. In intuition, standard diffusion SDEs with full-rank isotropic noise insertion are destructive of learning graph topology and feature representations. Theoretically speaking, for extremely sparse graphs (e.g. social networks, molecules) with low-rank adjacency matrices, the adjacency score functions are supported on a low-dimensional manifold embedded in the full adjacency matrix space. Thus, directly applying diffusion models on graph topology generation is not desirable: once the diffusion SDE is run in the full space of the adjacency matrix, lethal noise will be injected into the out-of-support regions and drives the signal-to-noise ratio to be essentially zero, which is fatal for training score networks.

Even for densely connected graphs, the standard diffusion model is problematic for topology generation. Unlike image pixels that are merely locally correlated, an adjacency matrix governs the message-passing pattern of the whole graph. Thus, isotropic Gaussian noise insertion severely distorts the message-passing pattern, by blindly encouraging message passing on sparsely connected parts, which impedes the representation learning of sparse regions.

1.1.4 Recommender Systems with Graph Neural Network

Recommender systems have been applied in many areas such as movies, music, news, social networks, online shopping, etc [63]. The recommendation task is to recommend one or a series of unobserved items to a given user.

The recommender systems focus on modeling the interactions between users and items, where the interactions can be view, click, purchase, etc. Early methods use singular value decomposition [174,175] and matrix factorization [104,112,148] to approximate the user-item interaction matrix. [171] shows that Matrix factorization can achieve high performance in general recommender systems. Recent methods mainly use deep learning techniques to model the user's preference and item properties. NeurMF [77] estimates users' and items' properties through Multi-layer perceptron (MLP) and generalized matrix factorization (GMF). ConvNCF [75] extends NeurMF with outer product and convolution operations. NAIS [76] uses the attention mechanism to recommend proper items based on user previous clicked items. Some methods apply auto-encoder [120] and utilize graph structure [220] to predict potential items.

Most recommendation data have implicit or explicit graph structure, for example, the interaction of user and items can be formulated as a user-item interaction graph. Therefore, graph-based models are naturally be applied to recommender systems, to model relationships between each entity, such as user-item interaction and user-user relationships. The graph-based recommendation model is one of the recommender system's research, as well as an important and widely applied application of the graph-based models.

The graph-based recommendation model is commonly used for the recommeder system with knowledge graphs or with sequential user behavior histories. A knowledge graph is a heterogeneous graph, where nodes function as entities, and edges represent relations between entities. Items and their attributes can be mapped into the KG to understand the mutual relations between items [63].

The user behavior histories contain the temporal information of the users' interest, which is widely used as the side information in recommender systems. In real-world applications, such as Amazon online shopping, the user behaviors are usually grouped as sessions, wherein each session, a user logins to the website, searches items, browses items, clicks items to view the item details and even purchases the items. The entire

process is categorized as a session, which can be further encoded as the graph structural data, since each user behavior in one session has strong correlations to other behaviors. Various methods have been proposed to power recommendations from the users' sessions perspective with graph-based models [158, 219, 222, 226, 242].

1.2 Motivation and Objectives

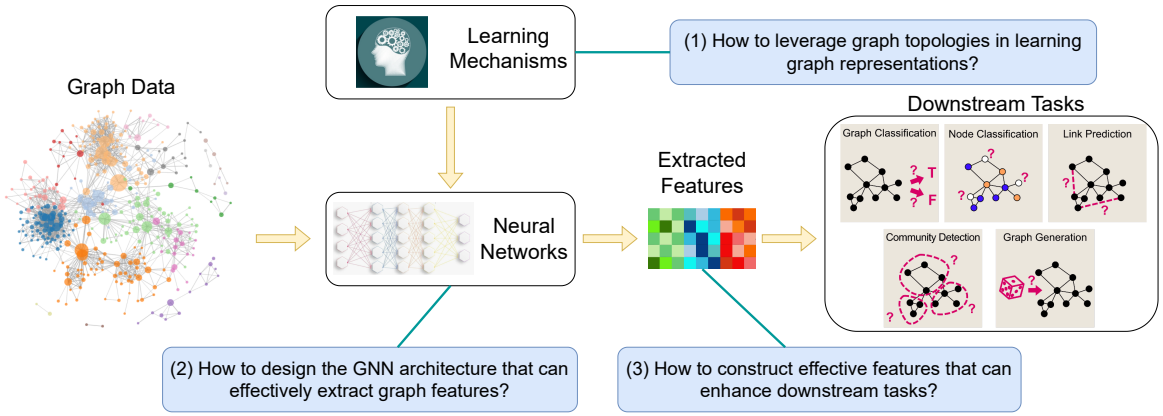


Fig. 1.1: An overview of the graph representation learning pipeline and the key research problems in the field.

The overall research goal of this thesis is to explore and develop improved methods for several critical steps in graph representation learning. As illustrated in Figure 1.1, graph representation learning based on deep learning methods requires designing graph neural networks (GNNs) that can effectively extract features from graph data. Additionally, powerful learning mechanisms are needed to enhance the GNNs' ability to utilize graph topology and extract graph features. Once the graph features are extracted, they can be applied to downstream tasks. However, a significant challenge in this process is that the features extracted directly by neural networks may be suitable for certain downstream tasks, such as graph and node classification, but may not be optimal for others, such as graph generation. Therefore, it is crucial to develop tailored feature extraction methods for specific downstream tasks.

Therefore, based on the graph representation learning pipeline, we have identified three main research problems: (1). How to leverage graph topologies in learning graph

representations? (2). How to design the GNN architecture that can effectively extract graph features? (3). How to construct effective features that can enhance downstream tasks? This section discusses the specific scenarios around the three aforementioned research problems that we are going to tackle, as well as the motivation and objectives.

Graph representation requires a thorough understanding of the graph topology, and one type of important method to tackle this problem is the contrastive learning method. In recent years, contrastive learning has been successfully applied to graph representation learning, leading to promising results [71, 211, 240]. These methods typically involve a data augmentation strategy to construct an augmented graph based on the input graph[1]. However, existing approaches have some limitations. Firstly, the construction of augmented graphs is stochastic, meaning that not all augmented graphs contribute equally to learning better representations based on the principle of contrastive learning [20,71,204]. Secondly, existing methods use a manually designed graph augmentation module with fixed parameters. This means that the augmentation module is not adaptively updated based on the historically generated augmented graphs. To address the limitations of graph contrastive learning aforementioned, a method using a meta-learning mechanism to generate suitable graph augmentations is introduced in this thesis. The method named Graph Meta-Contrast (GMeCo), is capable of facilitating effective graph contrastive learning.

Apart from the graph representation learning method, Graph Neural Networks are also essential in extracting valuable graph information. Most of the existing GNN models including GCN [100], GAT [201] and GraphSage [67] are low-pass filters that only perform low-pass filtering on feature vectors and do not have the non-linear manifold learning property [155]. They generate smooth node embeddings using low-resolution features, where neighboring graph nodes share similar graph features, and a local feature aggregation leads to informative representations. However, capturing the fine-grained graph details at high-resolution levels is also essential in learning graph representations. For instance, GNNs may fail on disassortative graphs [130, 160, 193], where locally connected nodes often exhibit evidently different features and labels. This heterogeneity emphasizes the necessity of using the high-pass graph filters to capture the disruptive

[1]Negative graphs are often sampled from the training dataset.

local patterns [130,160]. In this thesis, a multiresolution meta-framelet-based graph convolutional network (MMFGCN) is presented to tackle the problem in extracting graph features at different resolution scales.

Expanding our focus beyond understanding effective graph representations, we delve into the downstream tasks. Graph generation is a prominent task of graph representation learning, where effective graph features are learned by a graph generation model and brand-new graphs can be generated. In this thesis, we present a graph topology generation method that is based on the graph spectrum. Recently, a series of diffusion-based generative models have been proposed to overcome the limitations of likelihood-based models. Although being originally established for image generation [187], diffusion models exhibit great success in graph generation tasks with complex graph structural properties [91, 153]. Roughly speaking, diffusion is a mathematical technique that involves a Stochastic Differential Equation (SDE) used to smoothly transform real data into pure noise by adding more noise. Diffusion models are a class of probabilistic generative models that use this technique in reverse to generate representative data samples from noise. These models are trained to learn the underlying structure of a dataset by simulating how data points gradually become blurry due to added noise. Therefore, a well-trained diffusion model allows us to restore the original data by reversing the diffusion process and gradually removing the noise from the blurred sample. However, for graph-structured data, the standard diffusion model is problematic for topology generation. Unlike image pixels that are merely locally correlated, an adjacency matrix governs the message-passing pattern of the whole graph. Thus, isotropic Gaussian noise insertion severely distorts the message-passing pattern, by blindly encouraging message passing on sparsely connected parts, which impedes the representation learning of sparse regions. To tackle the graph topology generation problem, the Graph Spectral Diffusion Model (GSDM) is presented in this thesis. GSDM is driven by diffusion SDEs on both the node feature space and the graph spectrum space, effectively generating high-quality graph topology with significantly high accuracy and low cost.

We further study the application of GNNs on recommender systems. This thesis investigates the utilization of the GNN-based models in recommender systems and presents a novel recommendation model via multi-view GNN-Transformers to effectively tackle the feature extraction and message passing problems in the recommendation.

1.3 Major Contributions

As illustrated in Figure 1.1, the major research objective is to tackle the key problems in graph representation learning, and we have made significant progress in each of the key research problems. The major contributions of this thesis are summarized below.

- **Contribution 1**: This thesis introduces a novel graph representation learning model called Graph Meta-Contrast (GMeCo). GMeCo leverages graph data augmentation through contrastive learning and incorporates a meta-learning mechanism. GMeCo is specifically designed to mitigate the inductive bias introduced by the stochastic nature of augmented graph construction, thereby enhancing the efficacy of contrastive learning in graph representation learning. To validate the performance of our framework, comprehensive experiments were conducted, demonstrating that our proposed meta-learner can be effectively integrated into various Graph Neural Networks, resulting in enhanced performance. Additionally, this thesis offers an extensive comparison with state-of-the-art models across diverse datasets, further confirming the superiority and effectiveness of the GMeCo approach in graph representation learning.

- **Contribution 2**: This thesis introduces the Multiresolution Meta-Framelet Graph Convolutional Networks (MM-FGCN), a novel approach for multi-resolution analysis and feature extraction in graph structural data. This model, which extends beyond the learning algorithm GMeCo, is adept at extracting features from graphs at various resolution levels. A key component of this approach is the Multiresolution Meta-Framelet System (MMFS), a set of learnable multiresolution bases constructed using meta-band-pass filters. MMFS facilitates progressive resolution graph signal spaces characterized by denseness, tightness, and dilation and translation properties, allowing for effective multiresolution decomposition and reconstruction of any graph signal. Building upon the MMFS-based multiresolution transform, MM-FGCN is proposed for adaptive multiresolution graph signal processing. This method has been thoroughly tested, demonstrating state-of-the-art performance in comparison to other baseline methods, proving its efficacy in handling diverse graph datasets.

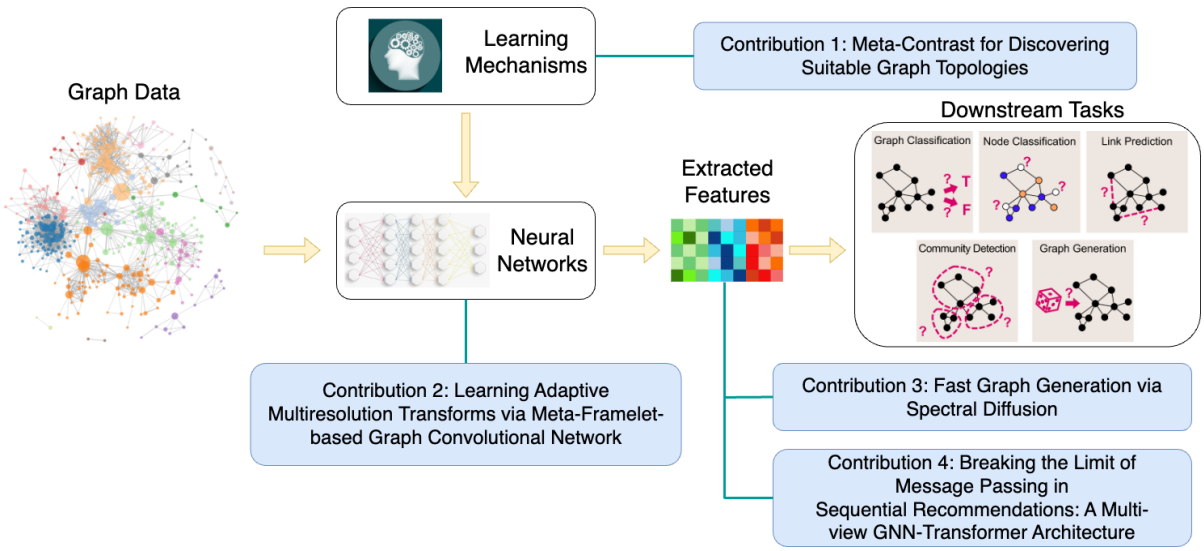


Fig. 1.2: An overview of the main contributions in this thesis.

- **Contribution 3**: For the graph generation task, an efficient approach based on graph spectral diffusion has been proposed, named Graph Spectral Diffusion Model (GSDM). GSDM addresses the limitations of traditional graph diffusion models by utilizing diffusion Stochastic Differential Equations (SDEs) in both node feature and graph spectrum spaces. Through stochastic analysis, GSDM is shown to offer a substantially stronger performance guarantee compared to standard graph diffusion models. Notably, it sharpens the reconstruction error bound from $\mathcal{O}(n^2 \exp(n^2))$ to $\mathcal{O}(n \exp(n))$, where n is the number of nodes. Evaluations on both synthetic and real-world graph generation tasks demonstrate that GSDM surpasses existing graph generative models in performance and achieves higher computational efficiency.

- **Contribution 4**: Based on the understanding of graph structural data and graph representation learning methods, this thesis delves into the application of graph-based methods in the realm of recommender systems, introducing a novel hierarchical graph aggregation model. One of the major advancements of this model is its efficient graph aggregation operations, making the model achieve significantly lower running complexity compared to conventional graph aggregation methods. The thesis further applies this model to develop an innovative sequential recommendation framework, demonstrating enhanced user behavior modeling capabilities over existing models. Additionally, a multi-view architecture is designed, and

Table 1.1: List of the commonly used notations.

Notations	Description
$\mathcal{G}$	The set of graphs
G	The graph data consists of $(\mathbf{X}, \mathbf{A})$
$\mathbf{X} \in \mathbb{R}^{n \times d}$	The node features
d	The feature dimension
$\mathbf{A} \in \mathbb{R}^{n \times n}$	The adjacency matrix
$\mathbf{Y}$	The set of labels
B	The batch of data in the training process
$\mathbf{D} \in \mathbb{R}^{n \times n}$	The graph degree matrix
$\mathbf{L} \in \mathbb{R}^{n \times n}$	The graph Laplacian matrix
$\Lambda \in \mathbb{R}^{n \times n}$	The diagonal eigenvalue matrix
$\mathbf{U} \in \mathbb{R}^{n \times n}$	The eigenvectors
θ	The neural network parameters

the Dirichlet sampling method is proposed to enhance the performance and robustness of the sequential recommendation model. Extensive experiments conducted on five real datasets, and subsequent comparisons to state-of-the-art sequential recommendation methods, reveal that our proposed model achieves superior performance relative to baseline methods.

An illustration of the four main contributions is summarized in Figure 1.2.

1.4 Notation List and Thesis Organization

The commonly used notations in this thesis are summarized in Table 1.1. This table provides a comprehensive list of symbols and their corresponding descriptions, which are utilized throughout the thesis to represent various elements such as graphs, node features, adjacency matrices, as well as elements during the training process such as labels, batch, and neural network parameters.

The remaining part of this thesis is organized as follows. In Chapter 2, the literature reviews about graph representation learning, Graph Neural Networks, graph generations, and graph-based recommender systems are presented. This aims to present a thorough overview of the related works and methods that are associated with the works presented in Chapter 3, Chapter 4, Chapter 5, and Chapter 6. In Chapter 3, a novel graph representation learning method: GMeCo is presented. Next, a novel graph neural network

that can extract graph features at different resolution scales is elaborated in Chapter 4. Chapter 5 focuses on elaborating a novel graph generation method GSDM, and Chapter 6 elaborates on the application of graph representation learning on recommender systems. Finally, the thesis is concluded in Chapter 7, with discussions on relevant future directions.

Chapter 2

Literature Review

2.1 Graph Representation Learning

The graph, composed of vertices and edges, serves as a versatile tool for modeling complex real-world systems, including social networks, protein-protein interactions, brain networks, traffic networks, physical interaction networks, knowledge graphs, and more. Graph representation learning aims to provide learning mechanisms for data with graph structures, to convert raw graph data into high dimensional vectors while preserving intrinsic graph properties [17]. Typical graph representation learning models include factorization models [104,112,148], random walks [62,163,196], graph kernels [11,107,181,235], deep neural networks [19,67,98,138,208], etc [17,68]. With learned graph representations, one can adopt them on various downstream tasks such as applications on community detection, recommendation, node clustering, node classification, edge/link prediction and clustering, and graph prediction and clustering.

In this section, we begin by discussing conventional kernel-based graph representation learning models, detailing their methodologies and applications. Following this, we delve into deep learning-based graph representation learning models, highlighting their advancements and the innovations in current emerging models. This comprehensive exploration will provide a thorough understanding of the evolution and current state of graph representation learning techniques.

2.1.1 Kernel-based Graph Representation Learning

One of the popular approaches to extract features from graph-structured data and learn its representations is to apply graph kernels, which are the functions to measure the similarity between graphs. The kernel methods refer to the machine learning algorithm that computes the similarity between pairs of data points. First, we give an introduction to kernel methods in machine learning, and then we elaborate on the graph kernel methods.

Kernel Methods: In the context of a non-empty data point set $\mathcal{X} \subseteq \mathbb{R}^d$, a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is considered a kernel on $\mathcal{X}$. A kernel k is defined as such if there exists a Hilbert space $\mathcal{H}_k$ and a feature map $\phi : \mathcal{X} \to \mathcal{H}_k$, satisfying $k(x, y) = \langle \phi(x), \phi(y) \rangle_{\mathcal{H}_k}$ for $x, y \in \mathcal{X}$, where $\langle \cdot, \cdot \rangle_{\mathcal{H}_k}$ represents the inner product in $\mathcal{H}_k$. The existence of feature map $\phi(\cdot)$ is contingent upon the positive semi-definiteness of k. An example of such a kernel is the inner product kernel, denoted as $k(x, y) = x^T y$.

The Gram matrix $K \in \mathbb{R}^{m \times m}$ plays a pivotal role in graph kernels and is defined for a finite set of data points $x_1, ..., x_m \in \mathcal{X}$, with $K_{i,j} = k(x_i, x_j)$ denoting the specific similarity measure between x_i and x_j for $i, j \in 1, ..., m$. If the positive semi-definiteness property holds for all possible sets of data points, then k qualifies as a valid kernel.

Weisfeiler-Lehman Kernel: The 1-dimensional Weisfeiler-Lehman (1-WL) or color refinement algorithm is a widely recognized heuristic for addressing the graph isomorphism problem [4]. The central idea behind the Weisfeiler-Lehman kernel involves aggregating information from a node's neighbors through a process of relabeling, where the target node is relabeled based on the labels of its neighbors.

The relabeling procedure operates as follows. Consider two graphs, denoted as G and H, and let $l : V(G) \cup V(H) \to \Sigma$ represent the observed vertex label function for both G and H. We refer to the relabeling iterations by the variable i, ranging from $i = 0$ to k. During each iteration of the 1-WL algorithm, we compute the new labeling function $l^i : V(G) \cup V(H) \to \Sigma$ as follows:

$$l^i(v) = \text{relabel}\left(\left(l^{i-1}(v), \text{sort}\left(\left\{\left\lfloor l^{i-1}(u) \mid u \in N(v)\right\}\right)\right)\right) \tag{2.1}$$

The Weisfeiler-Lehman kernel entails performing the above algorithm for iterations where $i \geq 0$. After each iteration i, a feature vector $\phi^i(G) \in \mathbb{R}^{\|\Sigma\|}$ is computed for each graph G.

Graphlet kernel. The graphlet kernel inherits the idea of the bag-of-words in text processing, i.e. the statistics of word occurrences without context reflect the properties of a document. In graphlet kernel, a similar idea called the bag-of-component is applied, by identifying the similarity of the components of two graphs, we can measure the similarity between the two graphs. The graphlet kernel method views graphs as a bag of graphlets, which are certain combination patterns of edges and vertices [108]. The method for constructing graphlets is detailed in [182].

Each graphlet represents an instance of an isomorphism type, which comprises a collection of graphs that are all isomorphic to one another. For example, this includes graphs with three vertices and two edges. Therefore, each graph can be decomposed into combinations of different isomorphism types, i.e. graphlets, and the graph can be viewed as a bag of graphlets. If two graphs contain the same number of each graphlet, we can infer that both graphs are highly similar.

Furthermore, to compute the similarity of two graphs G and H using graphlet kernels, one can compute the number of instances of isomorphism type of graphlets σ_i. Let $\phi(G)_{\sigma_i}$ denotes the number of instances of isomorphism type σ_i, $1 < i < N$, where N denotes the number of different types. Therefore, we can denote the feature map for graph G as

$$\phi_{\mathrm{GR}}(G) := (\phi(G)_{\sigma_1}, \ldots, \phi(G)_{\sigma_N}) .$$

Based on the feature map, the graphlet kernel k_{GR} for graph G and H can be computed as follows:

$$k_{GR}(G, H) = \langle \phi(G), \phi(H) \rangle \tag{2.2}$$

Shortest path kernel. The shortest path kernel assesses the attributes and lengths of the shortest paths between all pairs of vertices in two graphs [12]. For graphs G and H, the shortest path kernel is defined as follows:

$$k_{\mathrm{SP}}(G, H) = \sum_{\substack{(u,v) \in V(G)^2 \\ u \neq v}} \sum_{\substack{(w,z) \in V(H)^2 \\ w \neq z}} k((u, v), (w, z)), \tag{2.3}$$

where

$$k((u,v),(w,z)) = k_{\mathrm{L}}(l(u),l(w)) \cdot k_{\mathrm{L}}(l(v),l(z)) \cdot k_{\mathrm{D}}(d(u,v),d(w,z)) \tag{2.4}$$

k_L represents the kernel used for comparing vertex labels, while k_D serves as the kernel for comparing the shortest path distances.

Random walk kernel. Similar to the shortest path kernel, the random walk kernel assesses the similarity between two graphs by considering the sequences of vertices encountered during graph traversal. However, the random walk kernel is distinct in that it is based on random traversals. Proposed by [48, 93], the random walk kernel quantifies the number of shared walks (including label sequences) between two graphs. The greater the count of common walks, the higher the similarity between the two graphs.

2.1.2 GNN-based Graph Representation Learning

Graph Neural Net (GNN) based graph embedding techniques constitute the second category of graph embedding methods, utilizing GNNs to produce embeddings. These methods distinguish themselves from traditional approaches by exhibiting strong generalization capabilities for unseen nodes. Furthermore, GNN-based methods excel in leveraging node and edge attributes, as highlighted in [95].

Early research on graph representation learning applied multi-layer perceptron networks directly to acyclic graphs [189]. The idea of GNN was initially proposed by the pioneer works in [58] and further extended in the works [176]. Nowadays, abundant graph neural network models have been proposed, which also boost the development of the graph representation learning area. GNNs can be broadly classified into several categories: graph autoencoders (GAEs), convolutional graph neural networks (ConvGNNs), graph transformers, spatial-temporal graph neural networks (STGNNs), and recurrent graph neural networks (RecGNNs).

Convolutional Graph Neural Networks (ConvGNNs), sometimes referred to as Convolutional Graph Networks (GCNs), belong to a category of Graph Neural Network architectures that utilize convolutional operations to analyze and extract features from graphs. These models are inspired by Convolutional Neural Networks (CNNs) and are designed

to acquire hierarchical representations of non-uniform data. GCNs can be broadly classified into two main types: Spectral Graph Convolutional Networks and Spatial Graph Convolutional Networks.

Spectral Graph Convolution Operators. The conventional spectral graph convolution operator is determined by the graph Fourier transform. For a graph G, suppose x and y are two graph signals on G, and $\mathcal{F}$ and $\mathcal{F}^{-1}$ denotes the Fourier transform and inverse Fourier transform operators respectively. The spectral graph convolution on the graph signals $x *_G y$ is defined as follows:

$$\begin{aligned} x *_G y &= \mathcal{F}^{-1}(\mathcal{F}(x) \odot \mathcal{F}(y)) \\ &= \mathbf{U}(\mathbf{U}^T x \odot \mathbf{U}^T y) \\ &= \mathbf{U}\text{diag}(\mathbf{U}^T y)\mathbf{U}^T x, \end{aligned} \tag{2.5}$$

In this context, $\odot$ represents the element-wise Hadamard product, which means that it computes the product of corresponding elements in two matrices or vectors. $\mathbf{U}^T$ denotes the Fourier basis. With learnable network parameters f_θ, the spectral graph convolution can be reformulated as follows:

$$x *_G f_\theta = \mathbf{U}\mathbf{F}_\theta\mathbf{U}^T x.$$

Here, $\mathbf{F}_\theta$ represents a diagonal matrix comprising the trainable parameters.

Apart from the conventional graph convolution which is based on the graph Fourier transform, GNNs based on the Spectral Graph Wavelet Transform (SGWT) are another type of spectral graph convolutional networks. SGWT is determined by a graph wavelet generating kernel, which is denoted as $g : \mathbb{R}^+ \to \mathbb{R}^+$, and is parameterized by x_1 and x_2 which two positive real numbers that determine the transition regions, and two integers α and β. A valid instance of $g(\cdot)$ should satisfy the properties $g(0) = 0$ and $\lim_{x\to\infty} g(x) = 0$. The expression for $g(x; \alpha, \beta, x_1, x_2)$ is as follows:

$$g(x; \alpha, \beta, x_1, x_2) = \begin{cases} x_1^{-\alpha} x^\alpha & x < x_1 \\ s(x) & x_1 \le x \le x_2 \\ x^{-\beta} x_2^\beta & x > x_2 \end{cases}$$

Here, $s(x)$ is a cubic polynomial. The coefficients of $s(x)$ are settled by continuity constraints, such as $s(x_1) = s(x_2) = 1$, $s'(x_1) = \frac{\alpha}{x_1}$, and $s'(x_2) = -\frac{\beta}{x_2}$.

In SGWT, the spectral graph wavelet operator Ψ_g^s is determined based on a scaling parameter $s \in \mathbb{R}^+$ together with the graph wavelet generating kernel $g(\cdot)$, as $\Psi_a^s = \mathbf{U}g(s\mathbf{A})\mathbf{U}^T$, where $g(s\Lambda) = g(\text{diag}(s\lambda_1, \ldots, s\lambda_N))$. For a graph signal $\mathbf{x} \in \mathbb{R}^N$ on graph G, Ψ_a^s can be applied to filter the graph signal, resulting in $\mathcal{W}_{g,s}(\mathbf{x}) = \Psi_g^s\mathbf{x} \in \mathbb{R}^N$, where $\mathcal{W}_{g,s}$ denotes the wavelet transforms on graph.

The literature references [230] and [85] utilize Ψ_g^s to construct Graph Wavelet Neural Networks (GWNN) and graph scattering networks, respectively. Specifically, let $\Psi_a^{-s} \triangleq (\Psi_a^s)^{-1}$. The graph-wavelet-based convolution is defined as follows:

$$\mathbf{x} *_G \mathbf{y} = \Psi_g^{-s}\left(\Psi_g^s\mathbf{x} \otimes \Psi_g^s\mathbf{y}\right).$$

The GWNN comprises several graph-wavelet-based convolutional layers, and the operation of the l-th layer is specified as:

$$\mathbf{X}^{(l+1)}[:, j] = \rho\left(\sum_{k=1}^{d^{(l)}} \Psi_g^{-s}\Theta_{j,k}^{(l)}\Psi_g^s\mathbf{X}^{(l)}[:, k]\right).$$

Here, $\rho(\cdot)$ represents an activation function.

Spatial Graph Convolution Operators. Spatial-based Convolutional Graph Neural Networks (ConvGNNs) have a longer history than their spectral-based counterparts. In 2009, Micheli et al. [146] introduced the concept of addressing mutual dependencies in graphs by constructing non-recursive layers, while incorporating the concept of message passing from Recurrent Graph Neural Networks (RecGNNs).

Similar to how conventional Convolutional Neural Networks (CNNs) operate on images, spatial-based ConvGNNs define graph convolutions based on a node's spatial relationships. In this context, an image can be regarded as a specialized type of graph, where each pixel of the image corresponds to a node in the graph. Pixels are linked to their neighboring pixels. A filter can be used for a patch of neighbouring pixels to calculate the weighted average of pixel values among the central node and the neighboring nodes.

Similarly, spatial-based graph convolutions involve convolving the representation of the central node with the representations of its neighbors to update the central node's representation. From a different perspective, spatial-based ConvGNNs have a similar fundamental concept of message passing and information propagation with Recurrent

Graph Neural Networks (RecGNNs). The operation of spatial graph convolution essentially facilitates the propagation of node information along edges.

The Neural Network for Graphs (NN4G) [146], introduced in parallel with GNN, stands as one of the earliest works in the realm of spatial-based Convolutional Graph Neural Networks (ConvGNNs). Differing from Recurrent Graph Neural Networks (RecGNNs), NN4G explores mutual dependencies within graphs through a compositional neural architecture that employs independent parameters at each layer. The architecture allows for the incremental expansion of a node's neighborhood.

NN4G conducts graph convolutions by directly summing up a node's neighborhood information. Moreover, it integrates residual connections and skip connections to maintain information consistency across layers. Consequently, NN4G computes its next-layer node states using the following equation:

$$h_v^{(k)} = f(\mathbf{W}^{(k)^T} x_v + \sum_{i=1}^{k-1} \sum_{u \in N(v)} \Theta^{(k)^T} h_u^{(k-1)}), \tag{2.6}$$

where $f(\cdot)$ is an activation function.

Graph Auto-Encoders. Graph Autoencoders (GAEs) are unsupervised learning frameworks aimed at encoding nodes or entire graphs into a latent vector space and subsequently reconstructing graph data using this encoded information. GAEs serve multiple purposes, including learning network embeddings and graph generative distributions. In the context of network embedding, GAEs are employed to learn latent representations of nodes by reconstructing structural information from the graph, such as the graph's adjacency matrix. This enables the capture of essential features and relationships within the network. Regarding graph generation, various GAE methods employ different strategies. Some generate individual nodes and edges sequentially, while others generate an entire graph at once, offering flexibility in the modeling of graph structures.

Recurrent Graph Neural Networks. Recurrent Graph Neural Networks (RecGNNs) represent pioneering efforts in the field of Graph Neural Networks (GNNs). They focus on learning vertex representations using recurrent neural architectures, such as Recurrent Neural Networks (RNNs). The fundamental assumption of RecGNNs is that a node within a graph continuously exchanges information or messages with its neighbors until

a stable equilibrium state is achieved. For instance, in the model proposed by Scarselli et al [176], the model designs an information diffusion mechanism to update nodes' states by exchanging neighbour information recurrently until the hidden state of each node in the graph reaches a stable equilibrium. The hidden state of each node is updated in a recurrent manner as:

$$\mathbf{h}_v^{(t)} = \sum_{u \in \mathcal{N}(v)} f\left(\mathbf{x}_v, \mathbf{x}_{(v,u)}^{\mathbf{e}}, \mathbf{x}_u, \mathbf{h}_u^{(t-1)}\right) \tag{2.7}$$

in each iteration, where $f(\cdot)$ is a parametric function, $\mathbf{h}_v^{(t)}$ denotes the hidden state of node v at $t-$th iteration, and N(v) denotes the neighbours of node v. Through this iteration, the hidden state of each node can keep capturing the information from 1 to $t-$hop neighbours of each node.

Another classical RecGNNs model is the Gated Graph Neural Network (GGNN) [118] which adopts the recurrent neural networks (RNN) [60] as the backbone architecture, and iteratively aggregating information from the node neighbours through the gated recurrent unit (GRU). The iteration process can be shown as follows:

$$\mathbf{h}_v^{(t)} = GRU\left(\mathbf{h}_v^{(t-1)}, \sum_{u \in \mathcal{N}(v)} \mathbf{W}\mathbf{h}_u^{(t-1)}\right), \tag{2.8}$$

where $\mathbf{h}_v^{(t)}$ denotes the hidden state of node v at $t-$th iteration, and N(v) denotes the neighbours of node v.

Various graph Recurrent Neural Network (RNN) models, which are built upon the Long Short Term Memory (LSTM) architecture, have been introduced to extend the capabilities of vanilla LSTM models from sequential data to general graph-structured data [121, 161, 195].

Graph Transformers. Graph Transformers are effective at capturing long-range dependencies in dynamic graphs through an attention mechanism based on softmax, achieved by facilitating feature propagation within a single graph structure through message passing. A notable example of this is Graph-BERT [248], which is primarily designed as a pre-training method that exclusively relies on the graph attention mechanism, without incorporating any graph convolution operations. The core component of Graph-Bert

is a transformer encoder, denoted as $\mathbf{X}^{(l+1)} = \text{Softmax}(\frac{\mathbf{QK}^T}{\sqrt{d_h}})\mathbf{V}$, where $\mathbf{Q} = \mathbf{X}^{(l)}\mathbf{WQ}^{(l)}$, $\mathbf{K} = \mathbf{X}^{(l)}\mathbf{WK}^{(l)}$, and $\mathbf{V} = \mathbf{X}^{(l)}\mathbf{WV}^{(l)}$.

On the other hand, Graph2Seq [233] is a comprehensive end-to-end model designed for converting input graphs into sequences of vectors using an attention-based LSTM model. It comprises three key components: a graph encoder, a vertex attention mechanism, and a sequence decoder. The graph encoder encodes the node features and graph-level representations and outputs to the sequence decoder. The sequence encoder then computes the context vector sequence utilizing softmax-based attention.

2.1.3 Recent Emerging Methods

Contrastive learning provides a powerful self-supervised learning mechanism without labels, in which the methods learn the representation by a "contrastive loss" which can minimize the gap between similar data pairs while pushing apart dissimilar data pairs. Models applying contrastive learning have demonstrated effective advances in visual representation learning [20, 37, 38, 53, 202, 204, 246, 249, 250], natural language processing [42, 55, 132, 133, 184], graph representation learning [71, 88, 101, 165, 209, 211, 240]. Various loss functions [20,72,156] and view construction methods [204,218] for contrastive learning have also been studied.

In graph representation learning, the application of contrastive learning has been notably successful. Key to this approach is the generation of an *augmented graph* for each input graph, a process typically achieved through specific data augmentation strategies[1]. Hassani and Khasahmadi's (2020) method, for instance, transforms an input graph's adjacency matrix into a diffusion matrix to construct the augmented graph. The GraphCL framework [240] introduces four innovative data augmentation techniques, tailored for graph data. Despite these advancements, a critical limitation arises from the stochastic nature of augmented graph construction, which may not always align with the principles of effective contrastive learning [20, 71, 204].

Contrastive learning also significantly enhances the capabilities of graph-based models by leveraging self-generated positive and negative samples. Deep Graph Infomax

[1]Negative graphs are usually sampled from the training dataset.

(DGI) [211] extends the Deep Infomax model [80] to graph structures, contrasting the input graph with its negative samples generated through a corruption function. Similarly, InfoGraph [191] aims to maximize the mutual information between graph-level representations and their substructures. Graph Diffusion Networks (GDN) [101] innovatively combine spatial message passing with graph diffusion processes, effectively using diffusion as a denoising mechanism for message propagation through higher-order neighborhoods. Furthermore, MVGRL [71] integrates GDN with contrastive learning, contrasting graph data with diffusion graphs for enhanced representation. Lastly, GraphCL [240] diversifies the contrastive learning landscape by proposing four distinct graph data augmentation methods: node dropping, edge perturbation, attribute masking, and subgraph sampling.

2.2 Graph Generation

Graph generation models aim to generate graphs satisfying a quadruplet structure $G = (\mathcal{V}, \mathcal{E}, \mathbf{X}, \mathbf{E})$, where $\mathcal{V}$ is the vertex set, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the edge set, $\mathbf{X} \in \mathbb{R}^{N \times D}$ is the node feature matrix, $\mathbf{E} \in \mathbb{R}^{N \times N \times F}$ is the edge attributes, D and F denote the feature dimension.

In recent years, there have been several advanced graph generation strategies proposed, as outlined in [64, 141, 153, 186, 238, 245] and [91]. Among these, GraphRNN [238] and GraphVAE [186] generate nodes and edges sequentially with validity checks, while GAN-based models [30], VAE-based models [141], flow-based models [245], and score-based models [91, 153] generate the entire graph in an integrative way and exhibit high computational efficiency due to their node permutation-invariant property. Our proposed GSDM, which generates data by reversing a spectral diffusion SDE with a learned spectral score function, falls into the score-based model category. Specifically, GSDM accepts the destination of the reversed diffusion SDE as the final generated samples, without requiring any additional refinements.

A recently proposed score-based model, GDSS [91], is the first state-of-the-art diffusion-based generative model that simultaneously conducts nodes and edges generation. In essence, GDSS recasts the image diffusion paradigm [188] for graph generation. During the forward diffusion process, GDSS injects Gaussian noise into both the node features

and the adjacency matrix at each diffusion step. Then, a neural network is trained to learn the score function by minimizing the score-matching objective, which enables a reversion of graph data from noise via a reversed time diffusion process. However, such a directly borrowed diffusion model is incompatible with graph topology generation: unlike images that are feature-rich, the graph adjacency matrix is generally sparse and low-rank. Hence, injecting isotropic Gaussian noise into the sparsely connected parts of the adjacency matrix severely harms the graph data distribution and makes it hard to be recovered from Gaussian noise.

Representative methods like DiGress [212], and CDGS [83], represent the most recent state-of-the-art (SOTA) score-based graph generation models. Each of these models brings its unique approach to the task of generating graphs. DiGress employs a discrete diffusion process that iteratively modifies graphs by introducing noise. This process involves actions such as adding or removing edges and changing node categories. To simplify the problem of learning graph distributions, a graph transformer network is trained to reverse this diffusion process. This reversal involves a series of node and edge classification tasks. On the other hand, CDGS constructs a forward graph diffusion process that operates on both graph structures and inherent features using stochastic differential equations (SDE). These models contribute to the advancement of score-based graph generation by offering novel strategies and techniques to address the complexity of generating graphs with desired properties.

Recent advancements in the field of image generation and molecule generation have been exploring ideas that are relevant to our work. Subspace Diffusion [89] adopts image downsampling to restrict the image diffusion process via projections onto a sequence of progressively smaller subspaces as the data evolves towards noise. Wavelet-Score Diffusion (WSGM) [65] adopts wavelet transformation on the image to construct multi-level subspaces, and conducts diffusion training and sampling on the corresponding subspaces. In the field of molecule generation, Torsional diffusion [90], DiffDock [26], and FoldingDiff [224] explore diffusion method for 3D molecule generation with a similar principle: they utilize diffusions on the flexible reparameterized space of 3D molecules instead of canonical 3D Cartesian space. Corresponding reparameterized spaces include: internal angles of atoms [224], chirality tags, bond lengths and torsion angles of atoms [90], as well as translations and rotations of molecules [26].

2.3 Recommender Systems

2.3.1 General Recommendation Models

Recommender systems often contain graph-structured data and are associated with graph representation learning. In general, recommendation models focus on modeling the interaction between users and items, where the interaction can be view, click, purchase, etc. Previous models use singular value decomposition on user-item interaction matrix and produce low-rank approximation on the matrix by factorizing user and item embeddings [174, 175]. Rendle et al. [171] shows that Matrix factorization can achieve high performance in general recommender systems.

Recent methods apply deep learning models to construct user and item embeddings. NeuMF [77] estimates user and item's embedding by Multi-layer perception (MLP) and generalized matrix factorization (GMF). NAIS [76] applies attention network to approximate user embedding by his/her interacted items. ConvNCF [75] extends NeuMF [77] by replacing the inner product with the outer product and convolution operations. Mult-VAE [120] applies auto-encoder to reconstruct the user's clicked items.

2.3.2 Sequential Recommendation Models

The sequential recommendation is one of the main streams of recommender systems, which focuses on modeling user's sequential behaviors and making predictions/recommendations based on user historical behaviors, e.g. click, view, purchase, user-session, etc. The sequential recommendation is based on the assumption that historical behaviors can reflect the trend of the user's preference and therefore, we can recommend proper items not only based on the user's preference but also based on the user's recent status. For example, a user may want to buy a mobile phone case after purchasing a mobile phone, but will not buy a mobile phone after purchasing a model case, although both items are of the user's interest.

Previous works in sequential recommendation use the Markov chain to model user behavior patterns. For example, [170] uses matrix factorization and first-order Markov Chain to model users' global preferences and short-term interests. Recent methods model the users' sequential preferences with deep learning methods. [24, 78] apply recurrent

neural networks (RNN) to model the user behavior patterns by taking the user's clicked items into the RNN and outputting the prediction on the user's preference on the next item. SASRec [92] uses Transformer instead of RNN. Bert4Rec [192] adapt the famous Bert [34] architecture in NLP, replacing the single-directional attention in SASRec [92] to bi-directional attention architecture. Caser [198] adapts convolutional networks on user behavior sequences to predict the user's next action. HGN [140] adopts the hierarchical gating network for the next item prediction.

Some methods incorporate additional features, e.g. user profiles, item properties and reviews, to improve the recommendation [70, 79, 84, 115]. Wu et al. [225] apply user embedding to improve the performance of SASRec [92]. DIN [257] and DIEN [255] adopt user profiles and context features along with the user behavior histories. Ren et al. [168] apply generative adversarial networks with transformer architecture to process the item context information. Based on the transformer model, Li et al. [117] develop a time-interval aware self-attention model to incorporate time information into the sequential recommendation. ISSR [125] incorporates inter-sequence relations. In general, sequential recommendation models make predictions mainly based on the user behavior histories and can achieve state-of-the-art performance in nowadays recommendation models.

2.3.3 Graph-based Recommendation Models

Another stream of research in recommendation models is to exploit the user-item graph. Traditional methods such as ItemRank [59] apply a label propagation mechanism to directly propagate user-item scores through the graph, which prompts neighbour nodes to share similar labels. Recent methods apply deep learning techniques to learn the feature representations from the graph. GraphSage [67] introduces several graph aggregators to concentrate neighbouring information for each node and outputs the desired score. [208] combines attention mechanism with graph neural networks. Zhang et al. [247] introduce the gated attention network. Graph convolutional networks [98] also play an important role in the graph-based recommendation models. Developed from GCN [98], Gao et al. [47] apply graph CNN to model large graph. FastGCN [19] propose the fast graph convolutional networks to process the graph information. [74] further simplifies the GCN model while maintaining accuracy.

Instead of learning from the user-item bipartite graph, some works construct session graphs based on the user behaviors in certain sessions and apply graph-based models such as GCN, GraphSage or GAT to extract graph features and perform recommendations [158, 166, 219, 222, 226, 242]. Since the session-based recommendation usually involves item-correlation graphs extracted from short-term user behaviors, existing graph-based models are suitable to be applied in such tasks.

Chapter 3

Meta-Contrast for Discovering Suitable Graph Topologies

Contrastive learning has been recently applied to graph representation learning and obtained promising results. To make contrastive learning effective for representation learning on graphs, a key is to construct "desired" augmentations of each input graph as its positive samples such that along with negative samples (dissimilar graphs to the input graph) a contrastive learner can be trained to learn more robust and discriminative features. A "desired" augmented graph is expected to have the following properties: being similar to the input graph while containing differences to a certain extent to the input graph. However, existing contrastive methods on graph data fail to generate graph augmentations with the aforementioned properties. To address this issue, we propose a meta-learning framework for contrastive representation learning on graphs, where rather than using a handcrafted graph augmentation module a meta-learner is trained by converting the desired properties into constraints to generate augmented graphs adaptively. The whole meta-learning framework can be trained in an end-to-end manner. We conduct extensive experiments on several benchmark datasets compared with state-of-the-art baselines to demonstrate the superior performance of our proposed framework.

3.1 Motivation

Recently, the contrastive learning technique has been applied to graph representation learning and achieves some promising results [71, 211, 240, 261]. These methods often

involve a specific data augmentation strategy to construct a positive graph for any input graph[1], which is referred to an *augmented graph* of the input graph. For example, Hassani and Khasahmadi [71] propose to transform an adjacency matrix of an input graph to a diffusion matrix and use the diffusion matrix to construct a graph as the augmented graph. The GraphCL framework [240] proposes four different data augmentation methods imposing different priors for graph data. However, these methods have two main limitations. On the one hand, as the augmented graphs are *stochastically* constructed through a transformation from the input graph, some of them may not contribute to better representation learning based on the principle of contrastive learning [20, 71, 204]. For example, as shown in Fig. 3.1(d), if positive graphs are clustered, then the contrastive learning may result in a biased discriminative zone. If the augmented graphs are very similar to the input graph, as shown in Fig. 3.1(c), then they are not informative enough for representation learning, while if the augmented graphs are different to the input graph to a certain extent then they may become more similar to negative graphs than the input graph, as shown in Fig. 3.1(b). On the other hand, in these methods, the graph augmentation module is manually designed and its parameters are fixed during the whole contrastive learning procedure. In other words, the augmentation module fails to be adaptively updated based on historically generated augmented graphs.

To overcome the aforementioned limitations of graph contrastive learning, we propose to learn graph augmentations based on a meta-learning mechanism, as illustrated in Fig. 3.1(e). Specifically, we design a meta-learner that learns to generate suitable augmented graphs as positive samples given input graphs to conduct effective contrastive learning. The meta-learner takes a graph (of interest) as input, and generates an augmented graph. The augmented graph, a.k.a., the positive graph, along with the input graph and sampled negative graphs are fed into a graph representation learning model to learn high-level representations in a contrastive learning manner. The meta-learner is optimized based on two main objectives: 1) To maximize the mutual information (MI) of the representations between the generated augmented graph $\hat{G}$ and the input graph G, and 2) to keep $\text{MI}(\hat{G}, G)$ greater than $\text{MI}(\hat{G}, \tilde{G})$ for any $\tilde{G} \in \tilde{\mathcal{G}}$, where $\tilde{\mathcal{G}}$ is a set of sampled negative graphs. In this way, the meta-learner is expected to be able to generate a similar augmented graph with difference to some extent to an input graph.

[1]Negative graphs are usually sampled from the training dataset.

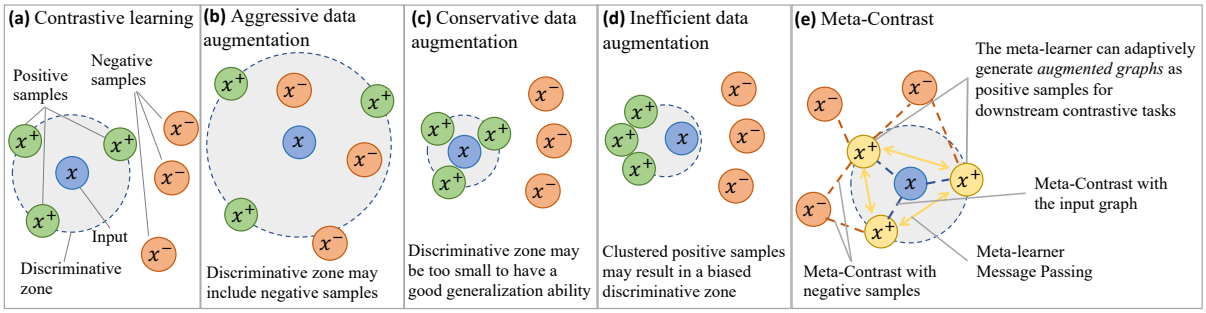


Fig. 3.1: An illustration of the motivation. (a) shows the basic logic of contrastive learning. (b-d) illustrate three scenarios when data augmentation fails. (e) describes our framework.

3.2 Methodology

Our proposed framework aims to learn self-supervised contrastive coding through a meta-learner to enable a backbone model[2] to conduct adaptive and effective contrastive learning, which further improves the performance on downstream graph prediction tasks. To be specific, the meta-learner learns to generate informative graph augmentation as a positive instance of any input graph. With the augmented "positive" graph, the input graph, and sampled negative graphs, one can apply any backbone model to learn more robust graph representations in a contrastive learning manner. In the sequel, we denote our proposed framework by GMeCo (**Me**ta **Co**nstrative learning for graph representation learning).

The model architecture is shown in Fig. 3.2. To begin with, we design a meta-learner $f_\omega : G \to \hat{G}$ to process any graph G in the dataset $\mathcal{G}$ to generate a desired positive augmentation $\hat{G}$. The overall objective of the meta-learner is 1) to reduce the similarity between G and $\hat{G}$ to some extent, i.e. $\min_\theta \text{sim}(f_\theta(G), f_\theta(\hat{G}))$ with the learned representation to introduce difference between G and $\hat{G}$.[3] Here, f_θ denotes the backbone model; 2) to ensure the augmented graph $\hat{G}$ is still "positive" to the input graph G, i.e. $\sup_{\tilde{G}} \text{sim}(\tilde{G}, \hat{G}) \leq \text{sim}(\hat{G}, G)$, where $\tilde{G}$ denotes the negative samples; and 3) to build up links between the newly generated graph and previously generated graphs. After we obtain the augmented graph $\hat{G}$ by the meta-learner, we input the tuple of graphs $(G, \hat{G}, \tilde{G})$

[2]Here, a backbone model is referred to as a graph representation learning model.

[3]For simplicity, we use $\text{sim}(G, \hat{G})$ to represent $\text{sim}(f_\theta(G), f_\theta(\hat{G}))$ in the following context.

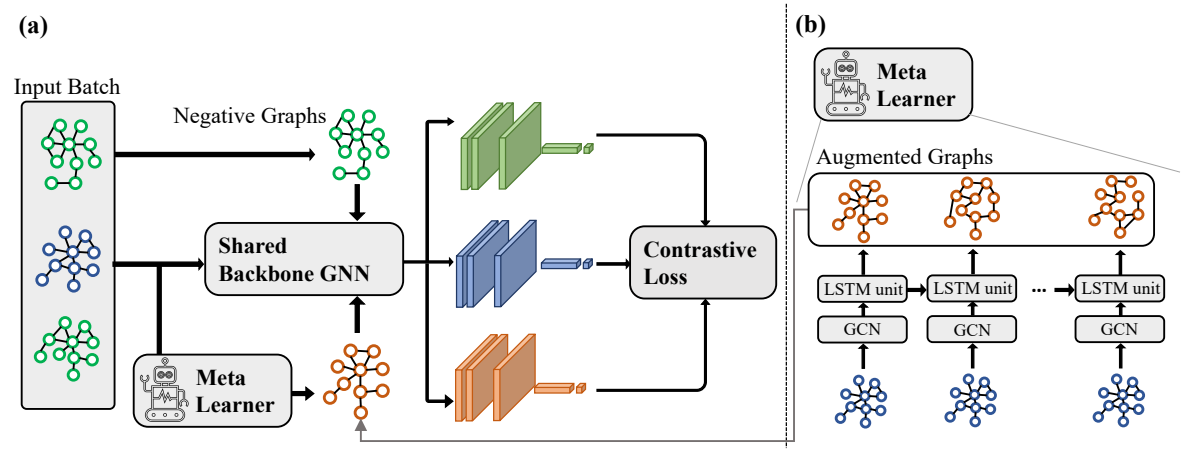


Fig. 3.2: Model Overview. (a): the overarching architecture of our proposed framework, GMeCo. (b): an example structure of the meat-learner: LSTM.

to the backbone model to learn representations by minimizing a contrastive loss based on the similarities $\text{sim}(G, \hat{G})$ and $\text{sim}(\tilde{G}, \hat{G})$. To enable the meta-learner to dynamically generate graph augmentations according to previous generations, we design the graph-LSTM-based model and graph-transformer-based model. We present the backbone model and the meta-learner in detail in the following sections.

3.2.1 Model Architecture

An input to the model consists of the adjacency matrix of the graph/sub-grpah $\mathbf{A} \in \mathbb{R}^{n \times n}$ and the initial node feature $\mathbf{X} \in \mathbb{R}^{n \times d}$. Similarly, we denote the augmented graph generated by the meta-learner as $\hat{G} = \{\hat{\mathbf{A}}, \mathbf{X}\}$, where $\hat{\mathbf{A}} \in \mathbb{R}^{n \times n}$ is the adjacency matrix of $\hat{G}$.

Regarding sampling negative graphs of the input instance G, for transductive learning (i.e. node classification task), in each batch B, we sample $|B|$ fixed-size sub-graphs, each with n nodes, from the main graph, as the input to our model. For inductive learning (i.e. graph classification task), in each batch, we sample multiple fixed-size graph $G \in B$ with each being an individual graph. In the batch, every two graphs are considered as a dissimilar (negative) pair, while each graph G and its augmented graph $\hat{G}$ are considered as a similar (positive) pair. The objective of the backbone model is to push apart the dissimilar pairs and pull together the similar pairs according to the local-global contrastive method [71, 211, 240].

Backbone networks. A backbone network encodes the graph information to latent features, i.e. $f_\theta(\mathbf{A}, \mathbf{X}) : \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d_h}$. We denote by $\mathbf{H} \in \mathbb{R}^{n\times d_h}$ the output node features of the backbone network. Similarly, by feeding the augmented graph $\hat{G}$ generated by the meta-learner to the backbone network, we obtain the corresponding node features $\hat{\mathbf{H}} = f_\theta(\hat{\mathbf{A}}, \mathbf{X})$, $\hat{\mathbf{H}} \in \mathbb{R}^{n\times d_h}$ for the augmented graph. To demonstrate the flexibility of the GMeCo framework, we adopt several commonly used graph encoder models as our backbone model $f_\theta(\mathbf{A}, \mathbf{X})$, including GCN [98], GAT [210] and Graph-u-net [46].

Graph contrastive learning methods typically rely on aggregating node features using a read-out function [211] or or nonlinear projections such as MLP [240] to generate graph features. These methods then compare node features with graph features either locally or globally. However, the global graph features obtained from the read-out function may not accurately capture the similarities between graphs. To address this limitation and better capture the similarity between pairs of graphs, we propose an optimal-transport-based contrastive learning approach. This method computes graph similarity based on the average of pairwise node similarities. It is worth noting that optimal transport or the Wasserstein measure has been used in other graph representation learning methods [17, 102, 199]. The optimal transport $\mathbf{T} \in \mathbb{R}_+^{n\times m}$ is defined as follows:

Definition 1 (optimal transport). Consider two discrete distributions $\boldsymbol{\mu} \in \mathbf{P}(\mathbb{X})$ and $\boldsymbol{\nu} \in \mathbf{P}(\mathbb{Y})$ which can be formulated as $\boldsymbol{\mu} = \sum_{i=1}^n \mathrm{u}_i \delta_{\boldsymbol{x}_i}$ and $\boldsymbol{\nu} = \sum_{j=1}^m \mathrm{v}_j \delta_{\boldsymbol{y}_j}$, respectively. Here $\delta_{\boldsymbol{x}}$ denotes the Dirac function centered at $\boldsymbol{x}$. The weight vectors $\mathbf{u} = \{\mathrm{u}_i\}_{i=1}^n$ and $\mathbf{v} = \{\mathrm{v}_i\}_{i=1}^m$ satisfy the constraint $\sum_{i=1}^n \mathrm{u}_i = \sum_{j=1}^m \mathrm{v}_j = 1$. Let $\Pi(\boldsymbol{\mu}, \boldsymbol{\nu})$ denote all the joint distributions of $\boldsymbol{x}$ and $\boldsymbol{y}$, with marginals $\boldsymbol{\mu}(\boldsymbol{x})$ and $\boldsymbol{\nu}(\boldsymbol{y})$. The optimal transport between the two discrete distributions $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ is defined as:

$$\begin{aligned} \mathbf{T}^* &= \operatorname*{argmin}_{\mathbf{T}\in\Pi(\boldsymbol{\mu},\boldsymbol{\nu})} \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathbf{T}}[c(\boldsymbol{x},\boldsymbol{y})] \\ &= \operatorname*{argmin}_{\mathbf{T}\in\Pi(\mathbf{u},\mathbf{v})} \sum_{i=1}^n \sum_{j=1}^m \mathbf{T}_{ij} \cdot c\left(\boldsymbol{x}_i, \boldsymbol{y}_j\right), \end{aligned} \tag{3.1}$$

where $c(\boldsymbol{x}_i, \boldsymbol{y}_i) = 1 - \frac{\boldsymbol{x}_i^T \boldsymbol{y}_i}{\|\boldsymbol{x}_i\|\|\boldsymbol{y}_i\|}$ is the cost function, which evaluates the cosine distance between $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$, and $\Pi(\mathbf{u}, \mathbf{v}) = \{\mathbf{T} | \mathbf{T}\mathbf{1}_m = \mathbf{u}, \mathbf{T}^T\mathbf{1}_n = \mathbf{v}\}$. The algorithm for computing $\mathbf{T}^*$ is presented in Algorithm 1.

Algorithm 1: Computing Optimal Transport.

Input: $\{h_i\}_{i=1}^n$,$\{h'_j\}_{j=1}^n$, β
$\boldsymbol{\sigma} = \frac{1}{n}\mathbf{1_n}$, $\mathbf{T}^{(1)} = \mathbf{1}\mathbf{1}^\top$
$\mathbf{C}_{ij} = c(h_i, h'_j)$, $\mathbf{O}_{ij} = \mathrm{e}^{-\frac{\mathbf{C}_{ij}}{\beta}}$
for $t = 1, 2, 3 \ldots$ **do**
  $\mathbf{Q} = \mathbf{O} \odot \mathbf{T}^{(t)}$ // $\odot$ is Hadamard product
  for $k = 1, 2, 3, \ldots K$ **do**
    $\boldsymbol{\delta} = \frac{1}{n\mathbf{Q}\boldsymbol{\sigma}}$, $\boldsymbol{\sigma} = \frac{1}{n\mathbf{Q}^\top\boldsymbol{\delta}}$
  end for
  $\mathbf{T}^{(t+1)} = \mathrm{diag}(\boldsymbol{\delta})\mathbf{Q}\mathrm{diag}(\boldsymbol{\sigma})$
end for
Return $\mathbf{T}$

We can consider the node feature representations $\mathbf{H}_G = \{\mathbf{h}_i\}_{i=1}^n$ and $\mathbf{H}_{G'} = \{\mathbf{h}'_j\}_{j=1}^n$ obtained from $f_\theta(G)$ and $f_\theta(G')$ as $\mathbf{u}$ and $\mathbf{v}$ respectively in (3.1). Based on this, we define the similarity between two graphs G and G' as:

$$\mathrm{sim}(G, G') = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n}\mathbf{T}^*_{ij} \cdot \frac{\mathbf{h}_i^T\mathbf{h}'_j}{\|\mathbf{h}_i\|\|\mathbf{h}'_j\|}, \tag{3.2}$$

where $\mathbf{T}^*$ represents the optimal node transport of the two graphs obtained from (3.1). It is important to mention that maximizing (3.2) leads to minimizing the Wasserstein distance between the two graphs.

Contrastive learning with the backbone model. Our objective is to maximize the mutual information, denoted by $\mathrm{MI}(f_\theta(G), f_\theta(\hat{G}))$, between the representations of input graph G and the augmented graph $\hat{G}$. To achieve this, we use the Jensen-Shannon estimator [154] as the basis for our loss function. By maximizing $\mathrm{MI}(f_\theta(G), f_\theta(\hat{G}))$ with our defined graph similarity measure in (3.2), the loss function can be expressed as:

$$\ell_{G,\hat{G}} = -\log\frac{\exp(\mathrm{sim}(G, \hat{G}))}{\exp(\mathrm{sim}(G, \hat{G})) + \sum\limits_{G' \in B, G' \neq G}\exp(\mathrm{sim}(G, G'))}, \tag{3.3}$$

where B represents the minibatch of training examples, and G''s are negative graphs of the input graph G in B. Additionally, we explore the use of different mutual information estimators, such as Jensen-Shannon divergence (JSD) [154], NT-XENT [20] and Donsker-Varadhan (DV) [40], in addition to the NCE loss in (3.3). This allows us to demonstrate the flexibility of our framework.

3.2.2 Meta-learner Networks

The meta-learner follows an encoder-decoder structure:

$$G = \{\mathbf{A}, \mathbf{X}\} \xrightarrow{\text{En}} S = f_\omega(\mathbf{A}, \mathbf{X}) \xrightarrow{\text{De}} \hat{\mathbf{A}} = \xi(\mathbf{S}) \longrightarrow \hat{G} = \{\hat{\mathbf{A}}, \mathbf{X}\},$$

where En and De are the encoder and the decoder, respectively, The encoder network is denoted by f_ω, with ω representing the learnable parameters. The encoder takes the node features $\mathbf{X}$ and the graph adjacency matrix $\mathbf{A}$ as inputs, and produces encoded node features $\mathbf{S} \in \mathbb{R}^{n \times d_m}$, where d_m is the dimension. We have designed several architectures for f_ω, including non-sequential architectures like MLP and GCN, as well as sequential architectures like GCN-LSTM and GCN-transformer. The decoder, $\xi(\mathbf{S}) = \sigma(\mathbf{S}\mathbf{S}^\top)$, is implemented by matrix multiplication followed by the ReLu activation function σ. The augmented graph of the input graph G consists of the generated adjacency matrix $\hat{\mathbf{A}}$ and the node feature $\mathbf{X}$, i.e. $\hat{G} = \{\hat{\mathbf{A}}, \mathbf{X}\}$. To train the meta-learner, we solve the following optimization problem:

$$\begin{aligned} &\min_{\omega} \frac{1}{|\mathcal{G}|} \sum_{G \in \mathcal{G}} \text{MI}(f_\theta(G), f_\theta(\hat{G})) \\ &\text{s.t. } \sup_{\tilde{G}} \text{MI}(f_\theta(\hat{G}), f_\theta(\tilde{G})) \leq \text{MI}(f_\theta(\hat{G}), f_\theta(G)) + \epsilon, \end{aligned} \tag{3.4}$$

where $\epsilon \geq 0$ and $|\mathcal{G}|$ is the number of training examples (graphs). Minimizing $\text{MI}(f_\theta(G), f_\theta(\hat{G}))$ can be achieved by minimizing the inverse of (3.2). Instead of explicitly sampling negative instances, we consider every G from the minibatch that is not used to generate $\hat{G}$ as the negative instances $\hat{G}$. To ensure that the generated augmented graph is a positive instance within the tolerance ϵ, we introduce the constraint $\sup_{\tilde{G}}\{\text{MI}(f_\theta(\tilde{G}), f_\theta(\hat{G}))\} \leq \text{MI}(f_\theta(\hat{G}), f_\theta(G)) + \epsilon$.

Lemma 3.1. *Based on the KKT conditions [109], the optimization problem in* (3.4) *can be written as*

$$\min_{\omega} \frac{1}{|\mathcal{G}|} \sum_{G \in \mathcal{G}} \Big[MI(f_\theta(G), f_\theta(\hat{G})) + \lambda \sup_{\tilde{G}} MI(f_\theta(\hat{G}), f_\theta(\tilde{G})) \Big], \tag{3.5}$$

where λ serves as a regularization coefficient that can adjust the weight of mutual information between negative samples and the augmented graphs.

Proof: By introducing a Lagrangian multiplier β, under the KKT condition, (3.5) can be rewritten as

$$\min_{\omega} \frac{1}{|\mathcal{G}|} \sum_{G \in \mathcal{G}} \Bigg[\mathrm{MI}(f_\theta(G), f_\theta(\hat{G})) + \beta \Big(\sup_{\tilde{G}} \mathrm{MI}(f_\theta(\hat{G}), f_\theta(\tilde{G})) - \mathrm{MI}(f_\theta(\hat{G}), f_\theta(\tilde{G})) \Big) \Bigg],$$

which can be further rewritten as

$$\min_{\omega} \frac{1}{|\mathcal{G}|} \sum_{G \in \mathcal{G}} \left[\mathrm{MI}(f_\theta(G), f_\theta(\hat{G})) + \frac{\beta}{(1-\beta)} \sup_{\tilde{G}} \mathrm{MI}(f_\theta(\hat{G}), f_\theta(\tilde{G})) \right].$$

where $\frac{\beta}{(1-\beta)}$ can be considered as the regularization coefficient that constrains the mutual information between the augmentations and the negative examples. By defining $\lambda = \frac{\beta}{(1-\beta)}$, we complete the proof.

In practice, we calculate $\sup_{\tilde{G}}\{\mathrm{MI}(f_\theta(\hat{G}), f_\theta(\tilde{G})\}$ by finding $\tilde{G}^* = \sup_{(\hat{G},\tilde{G}) \in B}\{\mathrm{MI}(f_\theta(\hat{G}), f_\theta(\tilde{G}))\}$, where B denotes a minibatch of examples. Thus, the loss function for minimizing $\mathrm{MI}(f_\theta(\hat{G}), f_\theta(\tilde{G}^*))$ through contrasting with other positive examples $G \in B$ is:

$$\ell^{\mathrm{meta}}_{\hat{G},\tilde{G}^*} = \log \frac{\exp(\mathrm{sim}(\hat{G}, \tilde{G}^*))}{\exp(\mathrm{sim}(\hat{G}, \tilde{G}^*)) + \sum_{G' \in B,\, G' \neq G} \exp(\mathrm{sim}(G', \tilde{G}^*))}, \tag{3.6}$$

where the meta-learner is updated in conjunction with the backbone networks in a minimax fashion. By combining (3.5) with the backbone model, we can express the overall learning objective of the GMeCo framework as follows:

$$\min_{\omega} \left(\max_{\theta} \frac{1}{|\mathcal{G}|} \sum_{G \in \mathcal{G}} \left[\mathrm{MI}\left(f_\theta(\hat{G}), f_\theta(G) \right) \right] + \frac{1}{|\mathcal{G}|} \sum_{G \in \mathcal{G}} \lambda \sup_{\tilde{G}} \{ \mathrm{MI}(f_\theta(\hat{G}), f_\theta(\tilde{G})) \} \right). \tag{3.7}$$

3.2.3 Speed-up for Large-scale Graph Datasets

For training with large-scale graph datasets, we have designed a speed-up method by replacing $\tilde{G}^*$ with randomly sampled negative data. In large-scale graph datasets like OGBG-PPA and OGBG-Code, each batch of data only represents a small fraction of the entire dataset. Consequently, the discriminative capability of our model when trained

Algorithm 2: GMeCo with GCN-LSTM.

input: *Batch size N, constant λ, the backbone model f_θ, the meta-learner f_ω (consists of the GCN module $f_{\omega-gcn}$, the LSTM module $f_{\omega-lstm}$, a dimension-reduction projection E and a dimension-raising projection D), the sequence length for LSTM $N_{\hat{G}}$, the initial hidden state h_0 and c_0.*

for *sampled minibatch* $B = \{g_k\}_{k=1}^{N}$ **do**

  for $k = 1$ *to* N **do**

    `// generate the augmented graph`

    $\{\mathbf{A}, \mathbf{X}\} = G_k$;

    $e = f_{\omega-\text{gcn}}(\mathbf{A}, \mathbf{X})$;

    $e_{\text{lstm}} = E(e)$;

    $h = h_0$, $c = c_0$;

    for $j = 1$ *to* $N_{\hat{G}}$ **do**

      $\mathbf{H}_j, h, c = f_{\omega-\text{lstm}}(e_{\text{lstm}}, h, c)$;

      $\mathbf{S}_j = D(\mathbf{H}_j)$;

      $\hat{\mathbf{A}}_j = \xi(\mathbf{S}_j)$;

      $\hat{G}_{k,j} = \{\hat{\mathbf{A}}_j, \mathbf{X}\}$;

    `// compute node representations of the input graph and its augmentation for computing MI`

    $h_i^{(k)} = f_\theta(G_k)$; `// the input graph`

    for $j = 1$ *to* $N_{\hat{G}}$ **do**

      $(\hat{h}_i^{(k,j)}) = f_\theta(\hat{G}_{k,j})$; `// the augmentation`

  for $k = 1$ *to* N **do**

    for $j = 1$ *to* $N_{\hat{G}}$ **do**

      if *Speed-up* **then**

        $\tilde{G}_{k,j}^* = \text{random}(G')$

      else

        $\tilde{G}_{k,j}^* = \sup_{G' \in B, G' \neq G_k} \{\text{MI}(f_\theta(\hat{G}_{k,j}), f_\theta(G')\}$;

      define $\ell_{k,j}^{\text{meta}}$ as $\ell_{\hat{G}_{k,j}, \tilde{G}_{k,j}^*}^{\text{meta}}$ in Eq (3.6);

      define $\ell_{k,j}$ as $\ell_{G_k, \hat{G}_{k,j}}$ in Eq (3.3);

  Update $f_{\omega-\text{gcn}}$, $f_{\omega-\text{lstm}}$, E and D to minimize $\frac{1}{N \times N_{\hat{G}}} \sum_{k=1}^{N} \sum_{j=1}^{N_{\hat{G}}} \ell_{k,j}^{\text{meta}}$;

  Update f_θ to minimize $\frac{1}{N \times N_{\hat{G}}} \sum_{k=1}^{N} \sum_{j=1}^{N_{\hat{G}}} \ell_{k,j}$;

return *backbone f_θ and meta-learner f_ω*

on random negative samples is not significantly different from when trained on $\tilde{G}^*$. By replacing $\tilde{G}^*$ with random negative samples from a batch, we can still guide the generated augment to the appropriate discriminative zone while simultaneously improving the

efficiency of the training phase.

We summarize GMeCo with a GCN-LSTM-based network as a meta-learner in Algorithm 2. In this algorithm, we utilize GCN layers to process the node features and subsequently employ an LSTM layer to generate a sequence of k augmented graphs, $\{\hat{G}_1, ..., \hat{G}_k\}$. Each augmented graph is then inputted into the backbone model, allowing us to independently calculate the corresponding loss. We use the average loss to update both the meta-learner and the backbone model accordingly.

3.2.4 Relation to Generative Adversarial Training

The entire GMeCo framework is trained using an adversarial approach. However, unlike GAN-based models such as GraphSGAN [35] and GraphCGAN [251], which generate a set of data examples following a learned distribution, our model directly generates appropriate augmentations for each input graph. To achieve this, GMeCo applies pair-wise contrastive loss, while GAN-based models primarily use discriminative loss to minimize the distribution gap between the generated domain and the domain of real data. This characteristic makes our model better suited for identifying proper discriminative zones for each individual graph example, as illustrated in Figure 3.1. The identification of proper discriminative zones is also a key distinction between our model and AD-GCL [194], another state-of-the-art method that generates augmentations for input graphs through a minimax optimization procedure. However, AD-GCL does not address the vanishing gradient problem commonly encountered in minimax optimization. In AD-GCL, the generator is always minimized throughout the entire training session, whereas our model maintains the generated augmentation within a certain range between the input data and the negative data, as specified in the constraint of (3.4). Additionally, JOAO v2 [239]employs adversarial training to enhance the performance of GraphCL [240] by providing adaptive augmentations. Unlike our method, the augmentations in JOAO v2 are limited to predefined spaces, similar to GraphCL. We empirically evaluate the performance of GMeCo compared to the aforementioned methods in Section 3.3.

3.3 Experiments

In this section, we evaluate GMeCo framework with state-of-the-art baselines on two tasks: *graph classification* and *node classification* on seven benchmark datasets. We also conduct extensive ablation studies to further illustrate the effectiveness of respective components in our model.

3.3.1 The GMeCo Architectures

Meta-learner. We test four types of network architecture as the meta-learner's encoder in the experiments:

1. **GCN-based meta-learner** consists of multiple GCN layers [98] to obtain the node features, where we test from 1 to 8 layers GCN.

2. **MLP-based meta-learner** contains a few MLP layers to process node features, where we test from 1 to 8 layers MLP.

3. **GCN-LSTM-based meta-learner** contains GCN + LSTM, where we use GCN layers to process the node features followed by an LSTM layer to generate a sequence of k augmented graphs $\{\hat{G}_1, ..., \hat{G}_k\}$. Each augmented graph is fed to the backbone model and the loss can be obtained independently. We calculate the average loss and then update the meta-leaner model accordingly.

4. **GCN-transformer-based meta-learner** is similar to GCN + LSTM, except we replace the LSTM units with a bi-directional transformer network.

Backbone networks. In the experiment, we adopt three commonly-used backbone networks, including GCN [98], GAT [210] and G-U-Net [46], for downstream classification tasks.

3.3.2 Benchmark and Datasets

We use three graph classification and four node classification benchmarks which are commonly used in the related works [19, 71, 98, 191, 211] to test the performance of our model with strong baseline methods [8, 31, 71, 100, 134, 150, 163, 210, 211, 223, 232, 237]. In node

Table 3.1: Statistics of graph benchmark datasets.

	Node Classification			*Graph Classification*			
	citeseer	**cora**	**pubmed**	**mutag**	**ptc-mr**	**imdb-b**	**imdb-m**
Graphs	1	1	1	188	344	1000	1500
Nodes	3,327	2,708	19,717	17.93	14.29	19.77	13.01
Edges	4,732	5,429	44,338	19.79	14.69	193.06	65.93
Classes	6	7	3	2	2	2	3

classification, we use the dataset CORA, Citeseer and Pubmed [177] to conduct experiments over five kernel methods, five supervised learning methods and six unsupervised methods. For graph classification task, we compare GMeCo with 13 supervised methods and six unsupervised methods, where we adopt the datasets: MUTAG [106], which contains mutagenic compounds; PTC [106], which contains compounds tested for carcinogenicity; IMDB-Binary (IMDB-B) and IMDB-Multi (IMDB-M) [235], which contain co-appearance (edge) of actors/actresses (node) from movies. We summarize the statistics of the benchmark datasets for graph classification and node classification in Table 3.1.

For the evaluation, we follow the common experiment procedures in the literature [71, 191, 209]. For node classification, we report the mean classification accuracy with standard deviation on the test nodes after 50 runs of training followed by a linear model. For graph classification, we report the mean 10-fold cross validation accuracy with standard deviation after 5 runs followed by a linear SVM. The linear classifier is trained by cross validation on training folds and the best mean classification accuracy is reported. We train our model using Adam optimizer [96] with learning rate 0.001, and the parameters are initialized following Xavier initialization [56].

3.3.3 Implementation Details

GMeCo with a GCN-LSTM-based meta-learner and a GCN backbone has been summarized in Algorithm 2. Here, we further elaborate more implementation details. After we obtain the embedding $\{e_i\}_{i=1}^{n}$ from the GCN module (i.e. $f_{\omega-\text{gcn}}$), where $e_i \in \mathbb{R}^{d_m}$

and n is the node number, we perform $e_{\text{lstm}} = \|_{i=1}^{n} [E(e_i)]$, where $E : \mathbb{R}^{d_m} \rightarrow \mathbb{R}^{d_{\text{lstm}}}$ is a learnable linear projection function and $\|_{i=1}^{n}$ denotes the concatenation of all node features. Thus, e_{lstm} is the concatenated node features being fed into the LSTM, and $n \cdot d_{\text{lstm}}$ is the hidden dimension in the LSTM encoder $f_{\omega-\text{lstm}}$. For calculation simplicity, we set $d_{\text{lstm}} < d_m$ as a dimension reduction procedure, and set $d_{\text{lstm}} = 64$ in our experiment. After we obtain the output from the LSTM encoder, we apply a learnable linear projection $D : \mathbb{R}^{d_{\text{lstm}}} \rightarrow \mathbb{R}^{d_{\text{m}}}$ to recover the hidden features for each node.

The network parameters of GMeCo are initialized with Xavier initialization [56]. We train our model using Adam optimizer [96] with a learning rate of 0.001. For GCN-based meta-learner, we follow the common settings in [71, 209] to vary the number of GCN layers from 1 to 8; for MLP-based meta-learner, we vary the number of MLP layers from 1 to 8; for GCN-LSTM-based meta-learner, we set vary the number of layer in the set of {2, 4, 6, 8}, followed by 1 LSTM layer; for GCN-transformer-based meta-learner, we vary the number of GCN layers in the set of {2, 4, 6, 8}, followed by 1 transformer layer. We set the sequence length for LSTM and transformer to be 5. For the backbone model, we follow the setting from the original GCN, GAT and graph-u-net models, respectively. For the GCN and GAT backbone models, we vary the numbers of the GCN and the GAT layers in the set of {2, 4, 6, 8}, respectively.

For graph classification, we vary the number of epochs, batch size, λ, and C in SVM in the set of {10, 20, 40, 80, 100}, {16, 32, 64}, {0.2, 0.4, 0.6, 0.8, 1.0} and {10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3}, respectively. For node classification, we set the number of epochs to 800, and the batch size in the set of {2, 4, 8}. Finally, we set the size of the hidden dimension of both node and graph representations to 512. For large-scale OGB dataset experiments, we adopt the speed-up method as elaborated in Section 3.4. The model is implemented with PyTorch and is run on a single GPU.

3.3.4 Experimental Results

The performance of our method compared with the benchmarks for graph classification and node classification are shown in Tables 3.2 and 3.3, respectively. The results show that the GMeCo framework achieves the best performance on most datasets. For graph classification, our method outperforms all the baseline models on MUTAG, IMDB-Binary,

Table 3.2: Experimental results on graph classification. Mean 10-fold cross validation accuracies for kernel, supervised, and unsupervised methods are reported. The best results within each category are given in boldface, whereas the global best is underlined.

	Method	MUTAG	PTC-MR	IMDB-B	IMDB-M
Kernel	SP [11]	85.2 ± 2.4	58.2 ± 2.4	55.6 ± 0.2	38.0 ± 0.3
	GK [181]	81.7 ± 2.1	57.3 ± 1.4	65.9 ± 1.0	43.9 ± 0.4
	WL [180]	80.7 ± 3.0	58.0 ± 0.5	**72.3 ± 3.4**	**47.0 ± 0.5**
	DGK [235]	87.4 ± 2.7	60.1 ± 2.6	67.0 ± 0.6	44.6 ± 0.5
	MLG [103]	**87.9 ± 1.6**	**63.3 ± 1.5**	66.6 ± 0.3	41.2 ± 0.0
supervised	GraphSAGE [67]	85.1 ± 7.6	63.9 ± 7.7	72.3 ± 5.3	50.9 ± 2.2
	GCN [100]	85.6 ± 5.8	64.2 ± 4.3	74.0 ± 3.4	51.9 ± 3.8
	GIN-0 [231]	**89.4 ± 5.6**	64.6 ± 7.0	**75.1 ± 5.1**	**52.3 ± 2.8**
	GIN-ϵ [231]	89.0 ± 6.0	63.7 ± 8.2	74.3 ± 5.1	52.1 ± 3.6
	GAT [210]	**89.4 ± 6.1**	**66.7 ± 5.1**	70.5 ± 2.3	47.8 ± 3.1
unsupervised	random walk [49]	83.7 ± 1.5	57.9 ± 1.3	50.7 ± 0.3	–
	node2vec [62]	72.6 ± 10.2	58.6 ± 8.0	–	–
	sub2vec [1]	61.1 ± 15.8	60.0 ± 6.4	55.3 ± 1.5	36.7 ± 0.8
	graph2vec [151]	83.2 ± 9.6	60.2 ± 6.9	71.1 ± 0.5	50.4 ± 0.9
	InfoGraph [191]	89.0 ± 1.1	61.7 ± 1.4	73.0 ± 0.9	49.7 ± 0.5
	mvgrl [71]	89.7 ± 1.1	62.5 ± 1.7	74.2 ± 0.7	51.2 ± 0.5
	GMeCo	**92.6 ± 1.1**	**65.3 ± 1.3**	**75.6 ± 0.7**	**53.4 ± 0.4**

and IMDB-Multi datasets with average accuracies 92.6%, 75.6% and 53.4%, 2.9%, 0.5% and 1.1% higher than the second best over all benchmarks, respectively. While on PTC-MR, we achieved the best accuracy of 65.3%, which is 2.8% higher than the second best in the unsupervised setting. For node classification, the proposed method obtains the best performance on both Citeseer and Pubmed datasets with 75.5% and 81.6% accuracies, margins of 2.2% and 1.5%, respectively in the unsupervised setting. We also achieve the second best on the Cora one. These results validate the effectiveness of our model. It is worth noting that compared to adversarial training methods such as GraphSGAN [35] and GraphCGAN [251], GMeCo achieves better performance for node classification.

Table 3.3: Experimental results on node classification. Mean node classification accuracy for supervised and unsupervised models. The input column highlights the data available to each model during training (**X**: features, **A**: adjacency matrix, **Y**: labels).

	Method	input	CORA	CITESEER	PUBMED
supervised	MLP [210]	**X**, **Y**	55.1	46.5	71.4
	ICA [134]	**A**, **Y**	75.1	69.1	73.9
	LP [260]	**A**, **Y**	68.0	45.3	63.0
	ManiReg [8]	**X**, **A**, **Y**	59.5	60.1	70.7
	SemiEmb [223]	**X**, **Y**	59.0	59.6	71.7
	Planetoid [237]	**X**, **Y**	75.7	64.7	77.2
	Chebyshev [31]	**X**, **A**, **Y**	81.2	69.8	74.4
	GCN [100]	**X**, **A**, **Y**	81.5	70.3	79.0
	MoNet [150]	**X**, **A**, **Y**	81.7 ± 0.5	–	78.8 ± 0.3
	JKNet [232]	**X**, **A**, **Y**	82.7 ± 0.4	73.0 ± 0.5	77.9 ± 0.4
	GAT [210]	**X**, **A**, **Y**	83.0 ± 0.7	72.5 ± 0.7	79.0 ± 0.3
	GraphSGAN [35]	**X**, **A**, **Y**	83.0 ± 1.3	73.1 ± 1.8	80.2 ± 2.0
	GraphCGAN [251]	**X**, **A**, **Y**	**84.0 ± 0.5**	**73.2 ± 1.0**	**80.7 ± 1.5**
unsupervised	Linear [211]	**X**	47.9 ± 0.4	49.3 ± 0.2	69.1 ± 0.3
	DeepWalk [163]	**X**, **A**	70.7 ± 0.6	51.4 ± 0.5	74.3 ± 0.9
	GAE [99]	**X**, **A**	71.5 ± 0.4	65.8 ± 0.4	72.1 ± 0.5
	VERSE [205]	**X**, **A**	72.5 ± 0.3	55.5 ± 0.4	–
	DGI [211]	**X**, **A**	82.3 ± 0.6	71.8 ± 0.7	76.8 ± 0.6
	MVGRL [71]	**X**, **A**	**86.8 ± 0.5**	73.3 ± 0.5	80.1 ± 0.7
	NWR-GAE [199]	X, A	81.8 ± 0.7	71.0 ± 0.7	78.3 ± 0.6
	GraphCL [240]	X, A	82.4 ± 0.7	73.1 ± 0.5	77.1 ± 0.6
	AD-GCL [194]	X, A	81.9 ± 0.7	72.2 ± 0.5	78.6 ± 0.5
	JOAO v2 [239]	X, A	82.9 ± 0.5	71.3 ± 0.6	78.5 ± 0.4
	LP-INFO [241]	X, A	82.7 ± 0.6	70.8 ± 0.6	78.5 ± 0.7
	GMeCo	**X**, **A**	85.3 ± 0.5	**75.5 ± 0.4**	**81.6 ± 0.5**

3.3.5 Ablation Study

To analyze the specific effect of each component in GMeCo framework, we conduct extensive ablation experiments and attempt to provide some explanations in this section. The corresponding experiment results are listed in Table 3.4. We try to answer the four research questions.

Table 3.4: Experimental results on ablation study. We vary one of components among backbone models, meta-learners and MI estimators and report the best performance of each variant.

		NODE CLASSIFICATION			*GRAPH CLASSIFICATION*			
		CORA	**CITESEER**	**PUBMED**	**MUTAG**	**PTC-MR**	**IMDB-B**	**IMDB-M**
Base	GCN	**85.27 ± 0.6**	**75.5 ± 0.4**	**81.62 ± 0.5**	**92.61 ± 1.1**	62.41 ± 1.7	**75.60 ± 0.7**	**53.43 ± 0.4**
	GAT	83.87 ± 0.4	73.58 ± 0.6	79.61 ± 0.5	91.94 ± 1.0	**65.25 ± 1.3**	73.41 ± 0.7	52.64 ± 0.4
	G-U-Net	84.03 ± 0.4	74.17 ± 0.7	80.38 ± 0.4	90.85 ± 1.1	63.60 ± 1.7	73.52 ± 0.6	52.31 ± 0.5
Meta	MLP	84.23 ± 0.4	74.14 ± 0.6	78.41 ± 0.6	90.74 ± 1.2	63.13 ± 1.6	73.45 ± 0.7	52.46 ± 0.6
	GCN	84.68 ± 0.4	74.41 ± 0.6	80.51 ± 0.5	91.52 ± 0.9	64.23 ± 1.4	73.11 ± 0.7	52.56 ± 0.6
	LSTM	85.08 ± 0.3	74.33 ± 0.6	80.95 ± 0.7	**92.61 ± 1.1**	**65.25 ± 1.3**	74.21 ± 0.7	**53.43 ± 0.4**
	Transformer	**85.27 ± 0.6**	**75.50 ± 0.4**	**81.62 ± 0.5**	91.91 ± 0.7	64.91 ± 1.1	**75.60 ± 0.7**	53.03 ± 0.6
MI Est.	NCE	83.62 ± 0.6	74.01 ± 0.7	79.21 ± 0.5	90.59 ± 1.1	63.16 ± 1.7	73.61 ± 0.8	52.58 ± 0.6
	JSD	**85.27 ± 0.6**	**75.50 ± 0.4**	**81.62 ± 0.5**	**92.61 ± 1.1**	**65.25 ± 1.3**	**75.60 ± 0.7**	**53.43 ± 0.4**
	NT-Xent	83.72 ± 0.4	74.32 ± 0.5	79.86 ± 0.6	91.62 ± 1.4	62.51 ± 1.2	73.83 ± 0.7	52.46 ± 0.6
	DV	83.48 ± 0.5	73.30 ± 0.6	80.21 ± 0.6	89.74 ± 1.5	62.67 ± 1.4	74.13 ± 0.6	52.02 ± 0.8
Sim	OT	**85.27 ± 0.5**	**75.50 ± 0.4**	**81.62 ± 0.5**	**92.61 ± 1.1**	**65.25 ± 1.3**	**75.60 ± 0.7**	**53.43 ± 0.4**
	Local-global	84.69 ± 0.5	74.67 ± 0.6	80.67 ± 0.4	91.57 ± 1.4	64.37 ± 1.5	74.12 ± 0.6	52.56 ± 0.5
	Global-global	82.57 ± 0.6	72.85 ± 0.5	79.89 ± 0.6	90.62 ± 1.3	62.55 ± 1.8	71.97 ± 0.6	50.81 ± 0.6

(1) Can GMeCo framework be applied to common GNN models? To test the compatibility of our GMeCo framework, we show the performance of several popular graph-based models: GCN, GAT, and G-U-Net with a Transformer meta-learner and JSD estimator for unsupervised contrastive learning. The results show that, all the backbone models can achieve relatively higher performance. For example, under the GMeCo framework, the GCN backbone achieves an average of 70.3% accuracy on Citeseer dataset, in contrast to 75.5% under supervised setting. The accuracy increases from 47.8% to 52.64% on IMDB-Multi for GAT. Therefore, the empirical results coincide with our assumption that the GMeCo framework does help improve the performance of common GNN models.

(2) What is the best architecture for the meta-learner? We design four types meta-learner architectures: MLP, GCN, GCN+LSTM, GCN+Transformer, as elaborated in section 3.2.2, combined with a GCN backbone and JSD estimator. The results show that, with consideration of the relationships between generated augmentations, the sequential architectures, transformer- and LSTM- based models achieve higher performance than the non-sequential GCN- or MLP-based meta-learners. In addition, the MLP meta-learner does not take into account the graph architecture, so that it achieves inferior performance than the GCN counterpart. However, MLP-based meta-learner achieves even

higher performance than most of existing unsupervised methods. For example, a combination of MLP+GCN+JSD achieves an average of 74.14% accuracy on Citeseer, which outperform all the baseline models in the unsupervised setting. This result confirms the strong competitiveness and effectiveness of the GMeCo framework.

(3) How does the MI estimator affect the performance? We conduct experiments using different MI estimators for estimating the mutual information between input graphs to augmented graphs and augmented graphs to negative samples. The tested MI estimator includes: InfoNCE [156], Jensen-Shannon (JSD) estimator [154], normalized temperature-scaled cross-entropy (NT-Xent) [20], and Donsker-Varadhan (DV) representation of the KL-divergence [40]. The results show that the commonly-used MI estimators are suitable for our meta-learning model. The JSD estimator can fit most of the cases, and achieve more than 1% improvements than other MI estimators.

(4) The effect of optimal-transport-based graph similarity estimation. We compare GMeCo framework (with optimal transport) with two common similarity estimation modes: local-global and global-global. In local-global mode, we adopt the method from [71, 211], where we contrast node features from one graph to the global feature of another graph with GMeCo framework. In global-global mode, we contrast the global features extracted from different graphs, where the mode is similar to [20]. The corresponding results in Table 3.4 reveal that the optimal-transport-based graph contrastive learning consistently outperforms the benchmark modes, and the optimal transport can be applied as a favorable metric to measure the graph similarities in self-supervised graph contrastive learning.

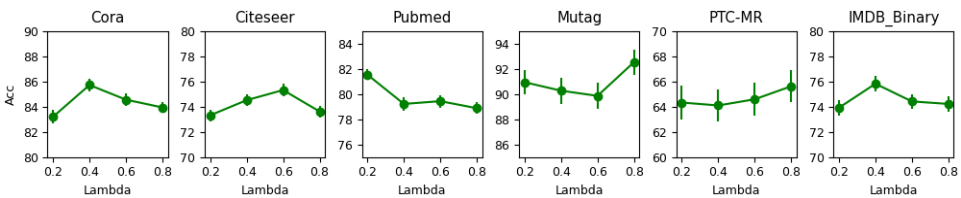
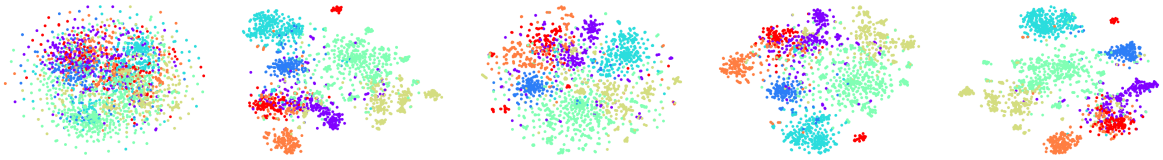


Fig. 3.3: Performance in terms of accuracy with different values of λ. A GCN+transformer+JSD model is adopted in GMeCo.

3.3.6 Hyper-parameter Tuning for λ

One important hyper-parameter in GMeCo is λ in (3.5), which determines the weight of pushing apart the input and augmented graphs. A smaller value of λ indicates that the augmented graph is more similar to negative samples and less similar to the input, while larger values of λ indicate the opposite. The choice of λ should aim to balance the similarity between the augmented graph and the input example. Figure 3.3 illustrates the performance of GMeCo with different values of λ. It can be observed that when λ is set between 0.2 and 0.8, the performance of GMeCo is satisfactory across different datasets.



(a) Raw features (b) MVGRL [71] (c) BGRL [200] (d) GraphCL [240] (e) GMeCo-LSTM

Fig. 3.4: t-SNE plot of the nodes representations from the Cora dataset.

3.4 Analysis on Effectiveness of GMeCo

To further evaluate the effectiveness of the GMeCo framework, it is practical to investigate the distribution of the graph/node representations learned by GMeCo. In this section, we analyze the distribution of the learned representations using t-SNE visualization and quantitative analysis to assess the alignment of the representations.

3.4.1 Distribution of Data Representations

Contrastive learning methods cluster similar data while pushing apart dissimilar data. To visualize the effectiveness of GMeCo on data clustering, we apply t-SNE [206] to project the embedding of node features, which are learned by our GMeCo framework as well as some strong state-of-the-art contrastive learning methods on the Cora dataset, to a 2D space. The visualization is shown in Figure 3.4. The results of t-SNE of the original node features are shown on the left-most side, and the learned representation of MVGRL,

BGRL, GraphCL and GMeCo-LSTM (GCN-LSTM meta-learner with GCN backbone) are shown respectively in the figure. In general, GMeCo can push apart data points from different classes, while clustering similar data points. Furthermore, compared to other methods, the clusters are more concentrated, and the boundaries between clusters are more clear, which potentially enhances the downstream classification task. Such phenomenon supports that the representations learned by GMeCo are more beneficial than the baselines on the node classification on Cora dataset, which corresponds to the results in Table 3.3.

3.4.2 Alignment Analysis

o assess the effectiveness of the meta-learner in GMeCo, we aim to examine the degree of alignment between the generated augmentations and the original graphs. Our objective, as stated in (3.4), is for the learned augmentations to exhibit higher mutual information with the original graphs compared to the mutual information between negative graph pairs. Consequently, we expect the generated augmentations to be more closely aligned with the original graphs compared to other negative examples. However, it is also expected that the generated augmentations may exhibit some misalignment with the original data when compared to other positive examples. To investigate the alignment of the representation between the real data and augmented data, we employ the alignment metric $\mathcal{L}_{align}$ as proposed in [218]. This alignment measure quantifies the expected distance between positive normalized graph/node pairs, and is defined as follows:

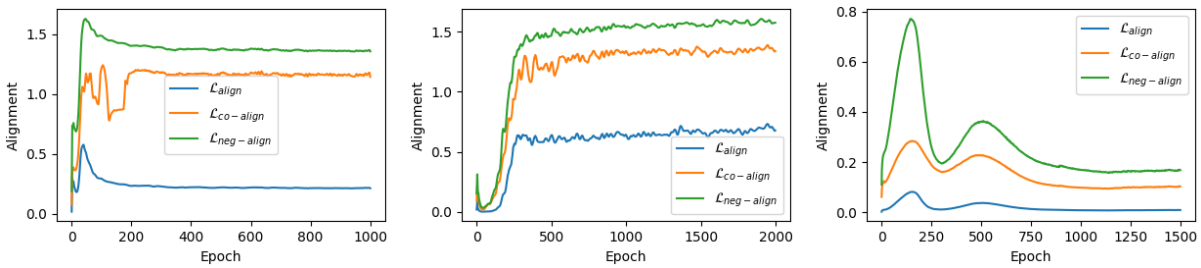
$$\mathcal{L}_{\text{align}}\left(f_\theta\right) \triangleq \mathbb{E}_{(x,x')\sim p_{\text{pos}}}\left[\left\|\frac{f_\theta(x)}{\|f_\theta(x)\|}-\frac{f_\theta(x')}{\|f_\theta(x')\|}\right\|_2^2\right], \tag{3.8}$$

where x and x' represent a random positive data pair sampled from p_{pos} that share the same label. To assess the efficacy of the augmentation, we introduce a co-alignment metric that quantifies the alignment between the real data and the augmented data:

$$\mathcal{L}_{\text{co-align}}\left(f_\theta\right) \triangleq \mathbb{E}_{(x,\hat{x})\sim p_{\text{pos}}}\left[\left\|\frac{f_\theta(x)}{\|f_\theta(x)\|}-\frac{f_\theta(\hat{x})}{\|f_\theta(\hat{x})\|}\right\|_2^2\right], \tag{3.9}$$

where $\hat{x}$ denotes a generated augmentation of x produced by the meta-learner. Likewise, we can measure the alignment of negative data pairs as

$$\mathcal{L}_{\text{neg-align}}\left(f_\theta\right) \triangleq \mathbb{E}_{(x,\tilde{x})\sim p_{\text{neg}}}\left[\left\|\frac{f_\theta(x)}{\|f_\theta(x)\|}-\frac{f_\theta(\tilde{x})}{\|f_\theta(\tilde{x})\|}\right\|_2^2\right], \tag{3.10}$$

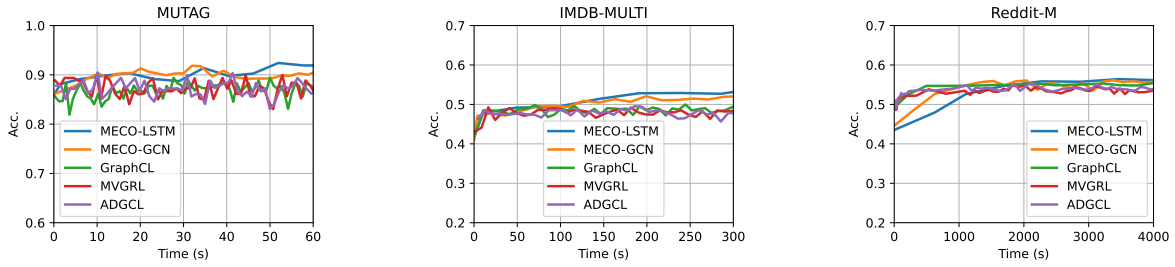


(a) Cora (Node) (b) PubMed (Node) (c) Mutag (Graph)

Fig. 3.5: Trajectories of $\mathcal{L}_{\text{align}}$, $\mathcal{L}_{\text{co-align}}$ and $\mathcal{L}_{\text{neg-align}}$ on Cora, Pubmed and Mutag datasets in training. The lower the value on the Y-axis, the better the alignment on the representation space.

where $(x, \tilde{x}) \sim p_{\text{neg}}$ represents a uniformly sampled negative pair of nodes/graphs. We present the alignment measures for GMeCo with GCN-LSTM meta-learner and GCN backbone during the training process on node classification tasks (Cora and PubMed) and a graph classification task (Mutag) in Figure 3.5. A smaller value of $\mathcal{L}_{\text{align}}$, $\mathcal{L}_{\text{co-align}}$ or $\mathcal{L}_{\text{neg-align}}$ indicates a better alignment between the corresponding node/graph pairs.

We can observe that, in general, GMeCo can learn node/graph representations such that the positive pairs are better aligned than the negative pairs. The augmented data aligns with the positive paired data better than the negative paired data, which corresponds to the positive intrinsic property of the augmentations (i.e. the constraint in (3.4). Meanwhile, the slightly less alignment to the positive data can help the backbone model to discover a suitable discriminative zone, which enhances the effectiveness of contrastive learning, as described in Figure 3.1.



(a) MUTAG (b) IMDB-M (c) Reddit-M

Fig. 3.6: Accuracy versus time for GMeCo-GCN and GMeCo-Lstm on Mutag, IMDB-Multi and Reddit-Multi datasets in training.

3.4.3 Runtime Analysis

We have plotted the training time versus testing accuracy (excluding evaluation time) in Figure 3.6 for two variants of GMeCo: 1. GCN meta-learner + GCN backbone; and 2. GCN-LSTM meta-learner (with LSTM length set to 5) + GCN backbone. The hidden dimension is set to 512, and the batch size is set to 16. For comparison, we have included GraphCL [240] and AD-GCL [194]. All models were trained on a single A100 GPU (40G). As depicted in 3.6 (a), the convergence rate is remarkably fast for small datasets like Mutag, with GMeCo-GCN and GMeCo-LSTM consistently outperforming the baselines after 10 seconds. For the medium-sized dataset IMDB-M, all methods converge within 200 seconds, and GMeCo-GCN and GMeCo-LSTM begin to outperform the others at around 100 seconds. For the large graph datasets, such as Reddit-M, GMeCo-GCN takes 10 minutes to achieve comparable results to the baselines, while GMeCo-LSTM takes 20 minutes. Both GMeCo-LSTM and GMeCo-GCN stabilize and consistently outperform the baselines after 30 minutes. Although our model's convergence rate is slightly slower than the baselines on large datasets, the application of our meta-contrast method leads to higher performance with acceptable increases in training time. Furthermore, the LSTM-based meta-learner requires additional time to compute a sequence of augmentations, which results in longer processing time per step compared to the GCN-based meta-learner. However, the LSTM-based meta-learner is still able to converge within a reasonable time frame. In the long run, the LSTM-based meta-learner generally outperforms the GCN-based meta-learner.

3.4.4 Comparison with Node Feature Generation

Instead of generating topological architectures for graphs, it is possible to generate augmented node features. However, GMeCo focuses on generating augmentations by constructing edges without generating node features. While many GNN methods adopt constructing augmentations based solely on topological structures, considering feature noise injection could potentially improve model performance. However, different graph datasets often have different types and quantities of node attributes. For example, Citeseer has 3,703 features per node, Cora has 1,433 features per node, while MUTAG only has 7 features. Learning to generate nodes with 3,703 features on a dataset like Citeseer,

Table 3.5: Experimental results on GMeCo's variant that generates both edges and node features. We apply the optimal transport method as the graph similarity measure. The values in the brackets indicate the increase/decrease of the performance compared to pure edge augmentation (i.e. results in Table 3.3.)

		NODE CLASSIFICATION			*GRAPH CLASSIFICATION*			
		CORA	**CITESEER**	**PUBMED**	**MUTAG**	**PTC-MR**	**IMDB-B**	**IMDB-M**
Base	GCN	83.32 (-1.95)	72.18 (-3.32)	79.1 (-2.52)	92.68 (+0.07)	62.32 (-0.09)	74.92 (-0.68)	52.93 (-0.50)
	GAT	79.84 (-4.03)	69.9 (-3.68)	81.67 (2.06)	88.83 (-3.11)	61.06 (-4.19)	75.04 (1.63)	51.47 (-1.17)
	G-U-Net	81.38 (-2.65)	69.86 (-4.31)	76.2 (-4.18)	90.99 (+0.14)	63.33 (-0.27)	74.17 (+0.65)	52.13 (-0.18)
Meta	MLP	78.8 (-5.43)	69.12 (-5.02)	78.98 (+0.57)	89.56 (-1.18)	59.33 (-3.80)	71.38 (-2.07)	51.29 (-1.17)
	GCN	82.6 (-2.08)	70.06 (-4.35)	80.05 (-0.46)	90.99 (-0.53)	61.41 (-2.82)	73.96 (+0.85)	52.23 (-0.33)
	LSTM	83.32 (-1.76)	71.28 (-3.05)	81.67 (+0.72)	92.47 (-0.14)	62.4 (-2.85)	74.23 (+0.02)	52.93 (-0.5)
	Transformer	83.02 (-2.25)	72.18 (-3.32)	80.82 (-0.80)	92.68 (0.77)	63.33 (-1.58)	75.04(-0.56)	52.85(-0.18)
MI Est.	NCE	81.26 (-2.36)	71 (-3.01)	78.36 (-0.85)	89.58 (-1.01)	59.7 (-3.46)	72.3 (-1.31)	51.6 (-0.98)
	JSD	83.32 (-1.95)	70.7 (-4.98)	81.67 (0.05)	92.68 +(0.07)	63.33 (-1.92)	74.9 (-0.70)	52.93 (-0.50)
	NT-Xent	80.22 (-3.50)	71.1 (-3.22)	78.34 (-1.52)	90.96 (-0.66)	60.69 (-1.82)	71.3 (-2.53)	51.53 (-0.93)
	DV	82.6 (-0.88)	72.18 (-1.12)	78.86 (-1.35)	90.41 (+0.67)	60.71 (-1.96)	75.04 (+0.91)	52.27 (+0.25)

which contains only 3,327 nodes, could be challenging. Additionally, different datasets typically have distinct types of features. For instance, Cora contains bag-of-words features, MUTAG contains atom-type features, and IMDB does not have any features. Therefore, learning feature noise injection through a fixed model architecture may not provide significant benefits for graph datasets in general.

To evaluate the impact of generating node features on GMeCo's performance, we introduce a modified version of the meta-learner model. This variant is capable of generating both topological structure and node attributes for augmentations. Learning attribute noise injection is analogous to directly learning node attributes, as the noise is obtained by subtracting the augmented attributes from the original attributes. The network structure is built upon the original meta-learner, with the addition of two MLP layers at the second last hidden layer to output the generated features of nodes. All other components remain unchanged. The results of this experiment are summarized in Table 3.5.

The experiment setup for node generation is identical to that of topology augmentation (refer to Table 3.3), and the hyperparameters are selected based on our experiment settings using the validation set. The results indicate that the performance of the node generation model on the core and Citeseer datasets is not as good as the original GMeCo architecture. This could be attributed to the high dimensionality of the attributes in both datasets, which might pose challenges for the meta-learner to effectively learn them. How-

ever, the model demonstrates promising results on the MUTAG and PTC-MR datasets, highlighting the effectiveness of learning to generate node features.

3.5 Conclusion

We propose a novel contrastive learning framework for graph representation learning. Our approach utilizes a meta-learner to generate augmented graphs for unsupervised contrastive graph code learning. We evaluate our framework on graph and node classification tasks and achieve superior prediction performance compared to SOTA methods. However, the training time for very large-scale graphs is slightly increased due to the introduction of the meta-learner. Future work will focus on improving the training efficiency of our framework on such graphs.

Chapter 4

Learning Adaptive Multiresolution Transforms via Meta-Framelet-based Graph Convolutional Network

4.1 Overview

In this chapter, we turn our attention from graph representation learning method to designing effective graph neural networks to enhance the graph feature extraction. The ubiquity of graph-structured data [45, 66, 173, 227, 258] in today's interconnected society has sparked immense interest in the machine learning community for processing and analysis of such data, which leverages mathematical representations like graphs to capture interdependencies between data entities. Graph neural network (GNN) has found widespread adoption due to ease of implementation and quality of prediction. Recent research [6, 52] underscores that most GNN models, including GCN [100], GAT [201], and GraphSage [67], fundamentally operate as low-pass filters in the context of graph signal processing [15]. They generate smooth node embeddings using low-resolution features, where neighboring graph nodes share similar graph features, and a local feature aggregation leads to informative representations.

However, capturing solely low-resolution information is insufficient for achieving a comprehensive graph representation. Low-resolution information represents graph signals that vary smoothly over the graph and are associated with low-frequency graph signals, whereas high-resolution information encompasses local disruption and detailed patterns that are associated with high-frequency graph signals. Thus, it is also crucial to capture the fine-grained graph details at high-resolution levels. For instance, GNNs may fail on disassortative graphs [130, 160, 193], where locally connected nodes often exhibit evidently different features and labels. This heterogeneity emphasizes the necessity of using the high-pass graph filters to capture the disruptive local patterns [130, 160]. In another example, for social network data, high- and low-frequency components represent micro and macro-level dynamics respectively. While the micro-level highlights individual interactions, revealing personal influences, the macro-level captures communities, clusters, and motifs, shedding light on broader social relations and group behaviors. Therefore, a GNN that relies solely on features from one or a few resolution levels fails to capture a comprehensive graph representation, necessitating the use of multiresolution graph analysis.

Recent advancements in multiresolution graph representation learning fall into two main categories, i.e. the 1) graph-structure-based approach [5, 52, 57, 234], which usually adopts down-sampling methods to partition the graph into multiple resolutions, or

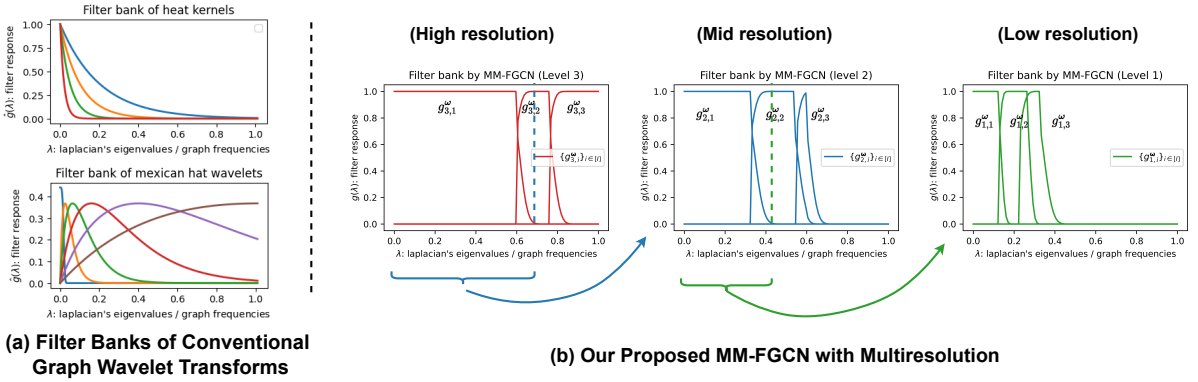


Fig. 4.1: Comparison of the filter banks of the conventional graph wavelet transforms with our proposed MM-FGCN with learnable multiresolution filter banks. We plot three levels of resolutions and each resolution level contains one low-pass filter and two high-pass filters.

adopt specially designed GNN such as Graph-U-Net [46] to capture the graph features at different resolutions. 2) Graph-spectral-based approach, where some of the methods under this category adopt low-pass and high-pass spectral filters [22,259]. Other methods adopt wavelet transforms [253, 254] to project graph signals to graph signal subspaces of different resolution levels. The wavelet frame transform provides an efficient way to obtain representations based on features of various scales.

Most current multiresolution methods rely on either heuristic, inflexible spatial down- and up-sampling strategies, or fixed, manually crafted spectral filters. For instance, the MR-GNN model [234] employs multi-hop convolution layers with receptive fields of a fixed size. UFGConv [253] and WFTG [39] leverage deliberately designed graph framelet transform to discern graph signals across various resolutions. Furthermore, PyGNN [52] utilizes a manually devised downsampling technique to categorize graph signals into different frequency levels. However, the reliance of these methods on fixed multiresolution analysis strategies imposes significant limitations on obtaining high-performing representations. In practice, various graph instances and distributions may manifest distinct scales and resolution factors, with their discriminative information residing at different resolution levels. Additionally, designing an appropriate multiresolution transform demands a deep understanding of the dataset-specific inductive bias, making it hard to generalize to other domains. Thus, employing fixed multiresolution analysis strategies fails to customize an appropriate multiresolution transform for individual graph instances.

To address this limitation, it is crucial to learn an adaptive multiresolution representation that can be automatically tailored to diverse graph instances and distributions. This motivates us to establish a meta-learner to generate the customized feature transform and multiresolution analysis strategy for each individual graph instance. In this chapter, we introduce the **M**ultiresolution **M**eta-**F**ramelet-based **G**raph **C**onvolution **N**etwork (**MM-FGCN**), a spectral-based method designed to learn adaptive multiresolution representations for different graph instances. For each input graph instance, the MM-FGCN first generates the meta-framelet generator, which consists of a set of customized band-pass filters in the frequency domain. The meta-framlet generator in turn induces a set of framelet-based multiresolution bases. Then, the input graph feature is decomposed into multiresolution components through projections onto each multiresolution basis. Finally, these multiresolution components are manipulated and passed to successive layers for downstream feature processing. As visualized in Figure 4.1, our MM-FGCN creates an adaptive multiresolution transform for each graph instance by learning a customized stratified multiresolution frequency partition in the frequency domain. In contrast, traditional filter-based and wavelet-based methods are confined to employing a fixed multiresolution analysis strategy across the entire graph dataset.

Contributions. In this chapter, we propose a novel MM-FGCN for adaptive multiresolution representation learning. The contribution of this chapter is three-fold.

- We introduce Multiresolution Meta-Framelet System (MMFS) (Section 4.3.1), a set of learnable multiresolution bases that can be simply constructed based on a set of meta-band-pass filters (Section 4.3.2).
- We show that MMFS induces a series of progressive resolution graph signal spaces that inherently possess denseness, tightness, and dilation and translation properties (Section 4.3.1). Thus, the multiresolution decomposition and reconstruction for any graph signal can be achieved by projections onto each basis in MMFS.
- Based on the MMFS-based multiresolution transform, we propose the Multiresolution Meta-Framelet-based Graph Convolutional Network (MM-FGCN) (Section 4.3.3) for adaptive multiresolution graph signal processing. Extensive experiments show that our model achieves state-of-the-art performance compared to other baseline methods.

4.2 Preliminary

We focus on undirected graphs represented as $G = (\mathbf{X}, \mathbf{A})$ with n nodes. Here, $\mathbf{X} \in \mathbb{R}^{n\times d}$ represents node features, and $\mathbf{A} \in \mathbb{R}^{n\times n}$ is the adjacency matrix, where $\mathbf{A}[i,j] > 0$ if an edge exists between node i and j, and $\mathbf{A}[i,j] = 0$ otherwise. The Laplacian matrix of the graph is defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D} = \mathrm{diag}(\mathbf{A}\mathbf{1}_n)$ is the diagonal degree matrix with $\mathbf{D}[i,i] = \sum_{i=1}^{n} \mathbf{A}[i,j]$, and $\mathbf{1}_n$ is an all-one vector of size n. Without specification, $\langle\cdot,\cdot\rangle$ denotes the inner product, $[n]$ denotes $\{1, ..., n\}$.

Graph Representation Learning. For any graph data G sampled from the graph domain $\mathcal{G}$, graph representation learning aims to learn a graph representation $f_{\boldsymbol{\theta}}(\cdot) : \mathcal{G} \mapsto \mathbb{R}^{n\times h}$, with which we can embed each node of G into a h-dimensional compact vector, facilitating downstream tasks such as node classification and graph classification. A desirable graph representation should be able to capture the essential graph structural properties.

Spectral Graph Signal Processing. A graph signal $x(\cdot)$ generally refers to a $G \mapsto \mathbb{R}$ mapping. As $x(\cdot)$ assigns a value to each of the n nodes, it is represented by a vector $\mathbf{x} \in \mathbb{R}^n$, where $\mathbf{x}[i]$ corresponds to the graph signal value assigned to the i-th node of G. In spectral graph signal processing [100, 185], the graph Laplacian $\mathbf{L}$ plays a crucial role in graph modeling and analysis, and it is tightly related to graph structural properties, including clusterability [23], connectivity [43], node distance, etc. In fact, $\mathbf{L}$ serves as a graph shift operator which enables us to transform a graph signal into the frequency domain and manipulate its frequency components. Suppose the eigendecomposition of the graph Laplacian is $\mathbf{L} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^\top$, the *graph spectrum* refers to the diagonal eigenvalue matrix $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1, ..., \lambda_n)$, and the *spectral bases* is the collection of eigenvectors $\mathbf{U} = (\mathbf{u}_1, ..., \mathbf{u}_n)$. Thus, a graph signal $\mathbf{x}$ can be transformed into the frequency domain via graph Fourier transform $\widehat{\mathbf{x}} = (\langle\mathbf{u}_1, \mathbf{x}\rangle, ..., \langle\mathbf{u}_n, \mathbf{x}\rangle)^\top = \mathbf{U}^\top\mathbf{x}$, and it can be reconstructed from its frequency components $\hat{\mathbf{x}}$ via the inverse graph Fourier transform $\mathbf{x} = \sum_i \langle\mathbf{u}_i, \mathbf{x}\rangle\ \mathbf{u}_i = \mathbf{U}\widehat{\mathbf{x}}$. Furthermore, one can apply a smooth filter $g_{\boldsymbol{\theta}}$ to manipulate frequency components of $\mathbf{x}$ by the *spectral convolution* [100]

$$g_{\boldsymbol{\theta}}(\mathbf{L}) * \mathbf{x} \triangleq \mathbf{U} g_{\boldsymbol{\theta}}(\boldsymbol{\Lambda})\mathbf{U}^\top\mathbf{x}.$$

In machine learning practice, applying spectral convolution to the graph feature $\mathbf{X}$ (which can be viewed as a d-dimensional graph signal) provides us with informative graph representation. Different implementations of filter $g_{\boldsymbol{\theta}}$ lead to desirable graph representations for different purposes, such as classification, denoising, smoothing, and abnormally detection [50, 116, 197].

Spectral Graph Multiresolution Analysis. Classic multiresolution analysis [25, 143] aims to decompose a signal into multiple components of varying resolutions, which can then be processed individually to provide a comprehensive representation of the signal. Let $L^2(\mathbb{R})$ be the measurable, square-integrable one-dimensional functions, where the inner product of $x, z \in L^2(\mathbb{R})$ is $\langle x, z\rangle = \int x(t)z(t)\mathrm{d}t$. Given a resolution factor $\gamma > 1$, the multiresolution decomposition for signals in $L^2(\mathbb{R})$ is determined by a series of progressive resolution function spaces $\{V_r\}_r$, where each V_r is a subspace of $L^2(\mathbb{R})$, and $V_r \subset V_{r'}$ if $r < r'$. The $\{V_r\}_r$ is expected to satisfy the *denseness*, *dilation property*, and *property* [144], ensuring that V_r collects the γ^r-resolution signals, and the multiresolution decomposition of any given signal x can be achieved by projecting it into each subspace V_r.

- *Denseness*: $\{V_r\}_r$ contains sufficient information to represent and reconstruct any signal, that is, the union of $\{V_r\}_r$ is dense in $L^2(\mathbb{R})$, and the intersection of $\{V_r\}_r$ is $\{0\}$.

- *Dilation property*: signals in V_r can be derived from signals in V_{r+1} by scaling them using a resolution factor of γ, that is, $\psi(t) \in V_r \iff D_\gamma\psi(t) = \psi(\gamma t) \in V_{r+1}$, where D_γ is the dilation operator.

- *Translation property*: when a signal x is translated for s in the spatial domain, its γ^r-resolution component translates for the same amount in the frequency domain, that is, $P_r(T_s x) = T_s P_r(x)$, where $P_r : L^2(\mathbb{R}) \mapsto V_r$ is the projection to V_r, and $T_s x(\cdot) = x(s - \cdot)$ is the translation operator.

The goal of multiresolution analysis is to determine a set of bases $\{\psi_{ri}\}_i$ that spans the desirable V_r, satisfying the denseness, dilation, and translation properties. Moreover, the γ^r-resolution component of a signal x should be derivable from its projection onto

each basis, i.e. $P_r(x) = \sum_i \langle \psi_{ri}, x \rangle \ \psi_{ri}$. Thus, the multiresolution decomposition of x can be achieved by $x = \sum_r P_r(x) = \sum_{r,i} \langle \psi_{ri}, x \rangle \ \psi_{ri}$. For instance, a proper choice of V_r is the collection of piecewise constant functions over $[-\gamma^r, \gamma^r]$, and ψ_{ri} can be set as the associated Haar-like wavelets [39].

For multiresolution graph analysis, one needs to extend the dilation and translation properties to the graph signal domain (where each graph signal is represented by a vector in $\mathbb{R}^n$) and determine the multiresolution graph bases $\{\boldsymbol{\varphi}_{ri}\}_{r,i} \subset \mathbb{R}^n$. To this end, one needs to define the spatial dilation and translation operators for graph signals by generalizing the scalar multiplication and node subtraction to the graph domain. According to the harmonic analysis theory [51, 190], the graph dilation and translation operators can be defined based on the graph Fourier transform. Consider a graph signal $\boldsymbol{\varphi} \in \mathbb{R}^n$ generated by a one-dimensional filter g, i.e. $\boldsymbol{\varphi} = \sum_k g(\lambda_k) \ \mathbf{u}_k$, the γ-dilation and v-translation of $\boldsymbol{\varphi}$ are defined as

$$D_\gamma \boldsymbol{\varphi} = \sum_k g(\gamma \lambda_k) \ \mathbf{u}_k, \ \forall \gamma > 0, \text{ and } T_v \boldsymbol{\varphi} = \sum_k g(\lambda_k) \mathbf{u}_k[v] \ \mathbf{u}_k, \ \forall v \in G,$$

respectively. Therefore, finding the desirable multiresolution bases is equivalent to identifying a set of filters $\{g_{ri}\}_{r,i}$ such that the bases $\{\boldsymbol{\varphi}_{riv}\}_{r,i,v}$ generated by $\boldsymbol{\varphi}_{riv} = \sum_k g_{ri}(\lambda_k) \mathbf{u}_k[v] \ \mathbf{u}_k$ satisfies the aforementioned conditions.

Finally, a desirable set $\{\boldsymbol{\varphi}_{riv}\}_{r,i,v}$ must exhibit *tightness*. The set of bases is called *tight* if and only if $\|\mathbf{x}\|^2 = \sum_{r,i,v} |\langle \boldsymbol{\varphi}_{riv}, \mathbf{x} \rangle|^2$ holds for arbitrary $\mathbf{x}$. Intuitively, tightness ensures that the projection operator onto these bases preserves the overall energy (norm) of the original graph signal. It's worth noting that this property, while essential, is less restrictive than orthogonality. As guaranteed by the polarization identity, it enables multiresolution decomposition via $\mathbf{x} = \sum_{r,i,v} \langle \boldsymbol{\varphi}_{riv}, \mathbf{x} \rangle \ \boldsymbol{\varphi}_{riv}$.

This decomposition can be equivalently expressed as $\mathbf{x} = \sum_{r,i,v} \boldsymbol{\varphi}_{riv} \boldsymbol{\varphi}_{riv}^\top \ \mathbf{x} = \boldsymbol{\Phi} \boldsymbol{\Phi}^\top \mathbf{x}$, where $\boldsymbol{\Phi}$ is an n-by-N *multiresolution transform matrix*, with each column representing a basis $\boldsymbol{\varphi}_{riv}$, and N is the total number of bases. As the multiresolution transform matrix is defined by concatenating the multiresolution bases, we will use these two terms interchangeably throughout the rest of this Chapter.

4.3 Methodology

Here, we propose the Multiresolution Meta-Framelet-based Graph Convolution Network (MM-FGCN), designed for adaptive multiresolution representation learning for varying graph instances. In Section 4.3.1 and Section 4.3.2, we construct a set of learnable multiresolution bases, termed Multiresolution Meta-Framelet System (MMFS). Our MMFS inherently possesses tightness, and it spans progressive multiresolution graph signal subspaces that satisfy denseness, dilation, and translation properties. For each graph instance, MM-FGCN first calculates the adaptive MMFS and the associated multiresolution transform matrix. This matrix enables us to decompose and manipulate the multiresolution components of the graph feature, yielding comprehensive graph representations (Section 4.3.3).

4.3.1 Multiresolution Meta-Framelet System

As mentioned in Section 4.1, learning the adaptive multiresolution bases is essential for obtaining a comprehensive graph representation. Suppose N is the total number of multiresolution bases, a straightforward approach is to learn the multiresolution transform matrix via a neural network $\mathcal{M}_{\boldsymbol{\xi}} : \mathcal{G} \mapsto \mathbb{R}^{n\times N}$ parameterized by $\boldsymbol{\xi}$, such that $\boldsymbol{\Phi} = \mathcal{M}_{\boldsymbol{\xi}}(\mathbf{X}, \mathbf{A})$. However, without additional constraints, this directly learned $\boldsymbol{\Phi}$ may fail to meet the tightness property $\boldsymbol{\Phi}\boldsymbol{\Phi}^\top = \mathbf{I}$, making the multiresolution decomposition infeasible. Even if we impose constraints on $\boldsymbol{\Phi}$ to ensure tightness, denseness, translation, and dilation properties, the constrained optimization process becomes challenging to solve due to numerical instability. Additionally, learning a dense $n \times N$ matrix requires an excessive amount of parameters, leading to a significant computational overhead.

To address these limitations, we construct a set of learnable multiresolution bases with much fewer parameters, called the Multiresolution Meta-Framelet System (MMFS). MMFS consists of a set of learnable graph framelets, each generated by a spectral *meta-filter*. Individually, these meta-filters are distinguished by their trainable bandwidth parameters and specific resolution levels, all while sharing a common trainable resolution factor. The following arguments show that our MMFS is born to be tight, and it spans

progressive multiresolution spaces that possess denseness, dilation, and translation properties. Hence, multiresolution decomposition can be achieved by using the MMFS-based multiresolution transform.

Definition 4.1 (Multiresolution Meta-Framelet System)**.** *Given the number of resolution levels $R > 0$, for the each resolution level $r \in [R]$, we define I spectral meta-filters $\left\{g^{\boldsymbol{\omega}}_{r,1}, ..., g^{\boldsymbol{\omega}}_{r,I}\right\}$. These meta-filters are mappings from the interval $[0, 1]$ to itself, and they are parameterized by a vector $\boldsymbol{\omega} \in \boldsymbol{\Omega}$. The collection of the $R \times I$ meta-filters is called the meta-framelet generator. We define the meta-framelet learner as $\mathcal{M}_{\boldsymbol{\xi}}(\cdot) : \mathcal{G} \mapsto \boldsymbol{\Omega} \times \mathbb{R}^+$, a neural network that maps any graph instance $G = (\mathbf{X}, \mathbf{A})$ to a specific meta-framelet generator $\boldsymbol{\omega}$ and a resolution factor γ. The Multiresolution Meta-Framelet System (MMFS) is defined as a set of graph signals $\{\boldsymbol{\varphi}_{riv}\}$, where*

$$\boldsymbol{\varphi}_{riv} = \sum_{k=1}^{n} g^{\boldsymbol{\omega}}_{r,i}\left(\gamma^{-J+r} \cdot \lambda_k\right) \mathbf{u}_k[v]\ \mathbf{u}_k, \tag{4.1}$$

where $(\boldsymbol{\omega}, \gamma) = \mathcal{M}_{\boldsymbol{\xi}}(\mathbf{X}, \mathbf{A})$, λ_k and $\mathbf{u}_k$ is the k-th eigenvalue and eigenvector of the graph Laplacian $\mathbf{L}$, and J is the smallest value such that $\gamma^{-J+R}\lambda_{\max}(\mathbf{L}) \leqslant 1$. The MMFS-based multiresolution transform matrix is defined as the concatenation of $\{\boldsymbol{\varphi}_{riv}\}$, that is

$$\boldsymbol{\Phi}_{\text{MM}} \triangleq \left(\mathbf{U}\ g^{\boldsymbol{\omega}}_{1,1}(\gamma^{-J+1}\boldsymbol{\Lambda})\ \mathbf{U}^\top, \cdots, \mathbf{U}\ g^{\boldsymbol{\omega}}_{R,I}(\gamma^{-J+R}\boldsymbol{\Lambda})\ \mathbf{U}^\top\right). \tag{4.2}$$

Definition 4.1 illustrates the construction of MMFS based on the meta-framelet generator. Here, $\boldsymbol{\varphi}_{riv}$ represents the basis comprising a r-resolution dilation and translation w.r.t the v-th node. At the r-resolution level, the meta-filter $g^{\boldsymbol{\omega}}_{r,i}$ filtrates the information localized around the v-th node. Notably, (4.2) enables the efficient computation of $\boldsymbol{\Phi}_{\text{MM}}$. This can be achieved by circumventing the need for eigen-decomposition of $\mathbf{L}$ through the application of Chebyshev approximation [32] to $g^{\boldsymbol{\omega}}_{r,i}(\gamma^{-J+r}\mathbf{L})$. The subsequent proposition offers a construction for the meta-framelet generator, ensuring that the MMFS meets the criteria of tightness, denseness, translation, and dilation. The proof is available in Section 4.5.

Proposition 4.1 (MMFS-based Multiresolution Decomposition)**.** *Following the notations in Definition 4.1, suppose the meta-framelet generator satisfies*

- $g_{1,1}^{\boldsymbol{\omega}}(\lambda)^2 + \cdots + g_{1,I}^{\boldsymbol{\omega}}(\lambda)^2 = 1,\ \forall\ \lambda \in [0,1]$.

- $g_{r,i}^{\boldsymbol{\omega}}(\gamma^{-J+r}\lambda) = g_{1,i}^{\boldsymbol{\omega}}(\gamma^{-J+r}\lambda)\ g_{1,1}^{\boldsymbol{\omega}}(\gamma^{-J+r-1}\lambda)\ \cdots\ g_{1,1}^{\boldsymbol{\omega}}(\gamma^{-J+1}\lambda),\ \forall r > 1,\ i \in [I]$,

then following the construction in Definition 4.1, the MMFS induced by $\{\boldsymbol{\varphi}_{riv}\}$ *forms a tight bases system. Here, the indices* (r,i,v) *are iterated over* $v \in [n]$*, with* (r,i) *drawn from the set* $([R]\times[I])\backslash(r,1) : 1 \leqslant r < R$*. For any graph signal* $\mathbf{x} \in \mathbb{R}^n$*, the multiresolution transform matrix is* $\boldsymbol{\Phi}_{\mathrm{MM}} \in \mathbb{R}^{n\times(R(I-1)n)}$*, the multiresolution decomposition is achieved by*

$$\mathbf{x} = \sum_{r,i,v} \langle \boldsymbol{\varphi}_{riv}, \mathbf{x}\rangle\ \boldsymbol{\varphi}_{riv} = \boldsymbol{\Phi}_{\mathrm{MM}}\boldsymbol{\Phi}_{\mathrm{MM}}^{\top}\ \mathbf{x},$$

where $\mathbf{x} \mapsto \boldsymbol{\Phi}_{\mathrm{MM}}^{\top}\mathbf{x}$ *is the multiresolution transform. Moreover, let* $V_r = \mathrm{span}(\{\boldsymbol{\varphi}_{riv}\}_{i,v})$*, the resulting subspaces* $\{V_r\}_r$ *turn out to be a series of progressive resolution space that possess denseness, dilation, and translation properties.*

Proposition 4.1 shows that, once the meta-filters of the 1-resolution level are determined, a desirable MMFS can be constructed in a stratified and iterative manner. As visualized in Figure 4.2, the r-resolution level meta-filters $\{g_{r,1}^{\boldsymbol{\omega}}, ..., g_{r,I}^{\boldsymbol{\omega}}\}$ induce a unitary partition within the support of $g_{r+1,1}^{\boldsymbol{\omega}}$, which is the low-pass filter of the $(r+1)$-resolution level.

4.3.2 Meta-Framelet Generator

To implement MMFS-based transform, the remaining step is to design the formulation of the meta-framelet generator $\left\{g_{1,1}^{\boldsymbol{\omega}}, ..., g_{1,I}^{\boldsymbol{\omega}}\right\}$ such that $\sum_i {g_{1,i}^{\boldsymbol{\omega}}}^2 \equiv 1$. This inspires us to simply set $\left\{g_{1,1}^{\boldsymbol{\omega}}, ..., g_{1,I}^{\boldsymbol{\omega}}\right\}$ as I band-pass filters to partition the $[0,1]$ interval into I regions. In this chapter, we instantiate each $g_{1,i}^{\boldsymbol{\omega}}$ as a meta-band-pass filter based on polynomial splines in [69], i.e.

$$g_{1,i}^{\boldsymbol{\omega}}(\lambda) \triangleq \begin{cases} 0, & \lambda \in [0, c_{i-1}-\varepsilon_{i-1}] \cup [c_i+\varepsilon_i, 1], \\ \sin(\frac{\pi(\lambda - c_{i-1}+\varepsilon_{i-1})}{4\varepsilon_{i-1}}), & \lambda \in (c_{i-1}-\varepsilon_{i-1}, c_{i-1}+\varepsilon_{i-1}), \\ 1, & \lambda \in [c_{i-1}+\varepsilon_{i-1}, c_i-\varepsilon_i], \\ \cos(\frac{\pi(\lambda - c_i+\varepsilon_i)}{4\varepsilon_i}), & \lambda \in (c_i-\varepsilon_i, c_i+\varepsilon_i), \end{cases} \tag{4.3}$$

where $\{c_1, \varepsilon_1, ..., c_{I-1}, \varepsilon_{I-1}\}$ are parameters encoded in $\boldsymbol{\omega}$. Specifically, for any $\boldsymbol{\omega} \in \boldsymbol{\Omega} \subset \mathbb{R}^{2(I-1)}$, we define

$$c_i \triangleq \frac{1}{\|\boldsymbol{\omega}\|^2} \sum_{j \leqslant i} \boldsymbol{\omega}[j]^2, \ \varepsilon_i \triangleq \alpha \min\{c_i - c_{i-1}, c_{i+1} - c_i\}, \tag{4.4}$$

where $\alpha \in (0, 1/2)$ is a predefined hyperparameter and it holds that $0 = c_0 \leqslant c_1 \leqslant \cdots \leqslant c_{I-1} \leqslant c_I = 1$. Notably, the parameterization of the meta-framelet generator uses only $2(I-1)$ parameters, significantly reducing the budget compared to the dense n-by-$(R \times (I-1) \times n)$ multiresolution transform matrices. Intuitively, the meta-filters adaptively decompose graph features into spectral channels and process frequency components at various resolution levels, leading to a flexible and comprehensive graph representation.

4.3.3 Multiresolution Meta-Framelet-based Graph Convolution Network

Leveraging the efficient construction and computation of MMFS-based multiresolution transform matrix $\boldsymbol{\Phi}_{\mathrm{MM}}$ in Proposition 4.1, we can now establish the Multiresolution Meta-Framelet-based Graph Convolution (MM-FGConv) and its associated graph pooling counterpart (MM-FGPool). These operators serve as meta-analogs to the conventional graph convolution and graph pooling methods [32]. The MM-FGPool operator is simply defined as $\mathrm{MMFGPool}_{\boldsymbol{\xi}}(\mathbf{H}; \mathbf{X}, \mathbf{A}) \triangleq \mathbf{1}^\top \boldsymbol{\Phi}_{\mathrm{MM}}^\top \mathbf{H}$, where the meta-framelet coefficients are aggregated and concatenated as the output of the readout of the final classifier. The computation of MM-FGConv is illustrated in Figure 4.2 and its details are presented in Algorithm 3.

An L-layer MMFS-based Graph Convolutional Network (MM-FGCN) is defined by

$$\mathrm{MMFGCN}_{\boldsymbol{\theta},\mathbf{w};\boldsymbol{\xi}}(\mathbf{A}, \mathbf{X}) \triangleq h \circ \mathrm{MMFGPool}_{\boldsymbol{\xi}} \left(\mathbf{H}_L \mathbf{W}_L; \mathbf{A}, \mathbf{X}\right), \tag{4.5}$$

$$\mathbf{H}_l \triangleq \sigma\left(\mathrm{MMFGConv}_{\boldsymbol{\Theta}_{l-1},\boldsymbol{\xi}}(\mathbf{H}_{l-1}; \mathbf{A}, \mathbf{X})\mathbf{W}_{l-1}\right), \ \forall\, l \in [L], \tag{4.6}$$

$$\boldsymbol{\theta} \triangleq \mathrm{vec}\left(\{\boldsymbol{\Theta}_l\}_{l\in[L]}\right), \ \mathbf{w} \triangleq \mathrm{vec}\left(\{\mathbf{W}_l\}_{l\in[L]}\right), \tag{4.7}$$

where $\mathrm{MetaFGConv}_{\boldsymbol{\Theta},\boldsymbol{\xi}}(\cdot)$ is the meta-framelet-based graph convolutional operator as defined in Algorithm 3, h is a fixed classifier (e.g. softmax), $\mathbf{W}_l \in \mathbb{R}^{d_{l-1} \times d_l}$ are learnable weight matrices, and $\sigma(\cdot)$ is the activation function. We define $(\boldsymbol{\theta}, \mathbf{w})$ as the base-

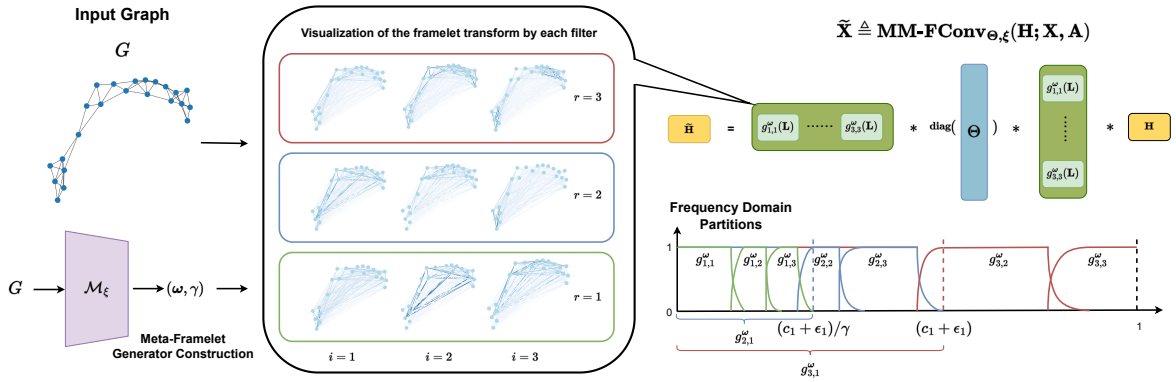


Fig. 4.2: the computation of MM-FGConv operator with a meta-framelet learner $\mathcal{M}_{\boldsymbol{\xi}}$ and learnable filter $\boldsymbol{\Theta}$.

parameters, and define $\boldsymbol{\xi}$ as the meta-parameters. By design, the MM-FGCN is permutational invariant [145], and is equipped with a learnable multiresolution transform that adapts to each graph instance.

Following the optimization paradigm introduced in MAML [44, 81], we employ meta-learning to train the MM-FGCN model. We aim to acquire a multiresolution transformation that enables the MM-FGCN backbone to adaptively and effectively represent individual graph instances. Specifically, our objective is

$$\min_{\boldsymbol{\theta}} \mathcal{L}_S(\boldsymbol{\theta}, \boldsymbol{\xi}^*(\boldsymbol{\theta})), \text{ s.t. } \boldsymbol{\xi}^*(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\xi}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\xi}), \tag{4.8}$$

$$\mathcal{L}_S(\boldsymbol{\theta}, \boldsymbol{\xi}) = \frac{1}{|S|} \sum_{(G,y)\in S} L(\text{MMFGCN}_{\boldsymbol{\theta},\mathbf{w};\boldsymbol{\xi}}(G), y), \tag{4.9}$$

where $L(\cdot,\cdot)$ is a loss function, e.g. the cross entropy. As outlined in Algorithm 4, we partition the training data into two sets: a meta-training set and a standard training set. In each iteration, we initiate an update to the meta-parameter $\boldsymbol{\xi}$, denoted as $\boldsymbol{\xi}'$, through gradient descent computed on a batch of meta-training data. Subsequently, we proceed to update all parameters $(\boldsymbol{\theta}, \mathbf{w}, \boldsymbol{\xi})$ using the full gradient evaluation at $(\boldsymbol{\theta}, \mathbf{w}, \boldsymbol{\xi}')$ based on the standard training data.

Algorithm 3: MM-FGConv

Input: graph data $G = (\mathbf{X}, \mathbf{A})$, graph Laplacian $\mathbf{L}$, the meta-framelet generators $\{g^{\boldsymbol{\omega}}_{r,i}\}$, a meta-neural network $\mathcal{M}_{\boldsymbol{\xi}}$, a learnable diagonal matrix $\boldsymbol{\Theta}$, the T-order Chebyshev polynomial approximation $\text{Chebyshev}_T(\cdot)$.
Output: $\widetilde{\mathbf{X}}$, the output processed graph signal.
$J \leftarrow \lceil \log_\gamma(\lambda_{\max}(\mathbf{L})) + R \rceil$
$(\boldsymbol{\omega}, \gamma) \leftarrow \mathcal{M}_{\boldsymbol{\xi}}(\mathbf{X}, \mathbf{A})$
$\widetilde{g}^{\boldsymbol{\omega}}_{r,i} \leftarrow \text{Chebyshev}_T(g^{\boldsymbol{\omega}}_{r,i})$, for $r \in [R]$, $i \in [I]$
$\boldsymbol{\Phi}_{\text{MM}} \leftarrow \Big(\widetilde{g}^{\boldsymbol{\omega}}_{1,1}(\gamma^{-J+1}\mathbf{L}), ..., \widetilde{g}^{\boldsymbol{\omega}}_{R,I}(\gamma^{-J+R}\mathbf{L})\Big)$
$\widetilde{\mathbf{X}} \leftarrow \boldsymbol{\Phi}_{\text{MM}} \boldsymbol{\Theta} \boldsymbol{\Phi}^\top_{\text{MM}}\ \mathbf{X}$
return $\widetilde{\mathbf{X}}$

Algorithm 4: Meta-training MM-FGCN

Input: graph dataset S, MM-FGCN parameters $(\boldsymbol{\theta}, \mathbf{w}; \boldsymbol{\xi})$, the empirical loss $\mathcal{L}(\cdot, \cdot)$. Learning rates $\beta_1, \beta_2 > 0$.
Output: optimized MM-FGCN $(\boldsymbol{\theta}^*, \mathbf{w}^*; \boldsymbol{\xi}^*)$
Split dataset $S_{\text{meta}}, S_{\text{main}} \leftarrow S$
for t in $[T]$ **do**
  $B \leftarrow \text{MiniBatch}(S_{\text{meta}})$
  $\boldsymbol{\xi}' \leftarrow \boldsymbol{\xi} - \beta_1 \nabla_{\boldsymbol{\xi}} \mathcal{L}_B(\boldsymbol{\theta}, \mathbf{w}; \boldsymbol{\xi})$
  $B' \leftarrow \text{MiniBatch}(S_{\text{main}})$
  $(\boldsymbol{\theta}, \mathbf{w}; \boldsymbol{\xi}) \leftarrow (\boldsymbol{\theta}, \mathbf{w}; \boldsymbol{\xi}) - \beta_2 \nabla \mathcal{L}_{B'}(\boldsymbol{\theta}, \mathbf{w}; \boldsymbol{\xi}')$
end for
$(\boldsymbol{\theta}^*, \mathbf{w}^*; \boldsymbol{\xi}^*) \leftarrow (\boldsymbol{\theta}, \mathbf{w}; \boldsymbol{\xi})$
return $(\boldsymbol{\theta}^*, \mathbf{w}^*; \boldsymbol{\xi}^*)$

Fig. 4.3: Left: the computation of MMFS-based multiresolution graph convolution operator. Right: implementation of MM-FGCN meta-training algorithm.

4.4 Experiments

4.4.1 Node Classification

Datasets. We conduct experiments on both assortative and disassortative graph datasets. A dataset is called assortative if its neighboring nodes usually have similar labels and features [142], as observed in citation networks and community networks. In contrast, disassortative datasets, such as co-occurrence networks and webpage linking networks, consist of numerous nodes with identical labels that are distant from one another. In this chapter, we evaluate the performance of our MM-FGCN on assortative datasets, including Cora, Citeseer, and Pubmed [178], as well as disassortative datasets, including Cornell [28], Texas [28], Wisconsin [28], Chameleon [172], and Squirrel [172]. For assortative datasets, following the configuration in [98], we allocate 20 nodes per class for training, 1,000 nodes for testing, and 500 for validation. As for disassortative datasets, we divide each dataset into training, validation, and test sets using a split ratio of 60%:20%:20%. All experimental results are averaged over 10 independent repetitions.

We show the statistics of the datasets for node classification in Table 4.1. For each

dataset, we list its graph statistics, data split, and homophily score, which is computed as follows.

Homophily. We adopt the homophily indicator $H(\mathcal{G})$ of the graph $\mathcal{G}$ from [128], which can be calculated as:

$$H(\mathcal{G}) = \frac{1}{|V|} \sum_{v \in V} \frac{| \{u : u \in \mathcal{N}(v) \text{ and } y(u) = y(v)\} |}{|\mathcal{N}(v)|},$$

where $| \{u : u \in \mathcal{N}(v) \text{ and } y(u) = y(v)\} |$ denotes the number of v' s directly connected nodes who have the same label as v and $|\mathcal{N}(v)|$ is the number of neighbouring nodes of v. Intuitively, high $H(\mathcal{G})$ indicates an assortative graph and vice versa.

Table 4.1: Statistics of the node-classification datasets used in our experiments. The homophily level of the dataset can be used to distinguish assortative and disassortative graph datasets.

Datasets	*Cora*	*Citeseer*	*Pubmed*	*Chameleon*	*Squirrel*	*Cornell*	*Texas*	*Wisconsin*
Homophily	0.83	0.71	0.79	0.25	0.22	0.11	0.06	0.16
Splits	140/500/1,000	120/500/1,000	60/500/1,000	60%/20%/20%	60%/20%/20%	60%/20%/20%	60%/20%/20%	60%/20%/20%
#Nodes	2,708	3,327	19,717	2,277	5,201	183	183	251
#Edges	5,429	4,732	44,338	36,101	217,073	295	309	499
#Features	1,433	3,703	500	2,325	2,089	1,703	1,703	1,703
#Classes	7	6	3	5	5	5	5	5

For the graph classification task, we adopt Mutagenicity, D&D, NCI1, Ogbg-molhiv, and QM7 datasets. The D&D and PROTEINS datasets are used for protein structure classification, which aims to categorize proteins into enzyme and non-enzyme structures. The NCI1 dataset is used for identifying chemical compounds that inhibit lung cancer cells. The Mutagenicity dataset is used for recognizing mutagenic molecular compounds that have the potential for drug development. The QM7 dataset is used for predicting the atomization energy value of molecules. The Ogbg-Molhiv is a molecular property prediction dataset for predicting whether a molecule inhibits HIV virus replication or not. All the datasets contain more than 1,000 graphs with varying graph structures (in terms of the average number of nodes and edges, the average degree of nodes) and node features. The statistics of each dataset are displayed in Table 4.2.

Baselines. We benchmark MM-FGCN against various competitive baselines on node classification tasks, including MLP, Chebyshev [33], GCN [98], Spectral CNN [13],

Table 4.2: Summary of the datasets for the graph property prediction tasks.

Datasets	PROTEINS	Mutagenicity	D&D	NCI1	ogbg-molhiv	QM7
# Graphs	1,113	4,337	1,178	4,110	41,127	7,165
Min # Nodes	4	4	30	3	2	4
Max # Nodes	620	417	5,748	111	222	23
Avg # Nodes	39	30	284	30	26	15
Avg # Edges	73	31	716	32	28	123
# Features	3	14	89	37	9	0
# Classes	2	2	2	2	2	1 (R)

GWNN [230], MPNN [54], GraphSAGE [67], LanczosNet [122], GAT [208], Non-Local GNN [129], Geom-GCN [160], two variants of UFGConv [253], i.e. UFGConvShrinkage and UFGConvRelu, and PyGNN [52]. We adhere to the original implementations of the baseline models as described in their respective papers.

Implementation details. We implement our model using PyTorch. We set the default number of filters as four, which is suitable for most of the datasets. The default Chebyshev approximation order is set to 6. The dimension of hidden variables is searched from $\{16, 32, 64\}$, and the level of filters are selected from $\{2, 3, 4, 5\}$. Other hyperparameters are set at: 0.001 for the learning rate, 0.001 for weight decay, 0.5 for dropout, and 2 for the number of MM-FGConv layers. These hyper-parameters are used for both node and graph classification tasks. For the graph classification task, we further apply our proposed MM-FGPool operation as elaborated in Section 4.3, followed by a linear classifier. For baseline methods in node classification, we adopt the code from the author's released implementation with the default settings. For graph classification, we adopt the experiment setting form [253], where we use two-layer GCN networks followed by the pooling methods listed in Table 4.4. Specifically, this experiment setting is comparable to our model's design, where we also adopt two convolutional layers and one pooling layer.

Results. As presented in Table 4.3, our proposed MM-FGCN model demonstrates state-of-the-art performance compared to all baseline models on both assortative and disassortative datasets. For disassortative datasets, compared to GCN, MM-FGCN achieves a significant performance gain of 34.7%, 25%, and 28.9% on the Cornel, Texas, and Wisconsin datasets, respectively. This evidence highlights that in disassortative datasets,

Table 4.3: Test accuracy (in percentage) for citation networks with standard deviation after ±. The results with the best performance are highlighted with *.

Method	Assortative			Disassortative				
	Cora	Citeseer	Pubmed	Cornell	Texas	Wisconsin	Chameleon	Squirrel
MLP	55.1 ± 1.1	46.5 ± 1.3	71.4 ± 0.7	81.6 ± 6.3	81.3 ± 7.1	84.9 ± 5.3	48.5 ± 3.0	31.5 ± 1.4
SPECTRAL [13]	73.3 ± 1.4	58.9 ± 0.8	73.9 ± 0.6	52.1 ± 7.2	57.5 ± 7.5	56.7 ± 5.1	62.4 ± 2.4	53.8 ± 2.2
CHEBYSHEV [33]	81.2 ± 1.2	69.8 ± 0.9	74.4 ± 0.6	53.1 ± 7.8	59.1 ± 7.2	55.3 ± 5.3	60.2 ± 2.9	55.4 ± 2.5
GWNN [230]	82.8 ± 0.9	71.7 ± 1.1	79.1 ± 0.8	56.8 ± 7.6	63.1 ± 6.9	61.2 ± 4.9	63.7 ± 2.8	55.4 ± 2.3
MPNN [54]	78.0 ± 1.1	64.0 ± 1.9	75.6 ± 1.0	52.3 ± 7.0	58.2 ± 5.4	56.4 ± 5.1	60.8 ± 2.7	53.1 ± 2.3
GRAPHSAGE [67]	74.5 ± 0.8	67.2 ± 1.0	76.8 ± 0.6	54.2 ± 7.8	60.5 ± 7.2	58.7 ± 5.3	62.4 ± 2.9	55.4 ± 2.5
LANCZOSNET [122]	79.5 ± 1.8	66.2 ± 1.9	78.3 ± 0.3	53.1 ± 7.5	60.4 ± 7.2	57.1 ± 4.7	65.2 ± 2.5	54.1 ± 2.1
GCN [98]	81.5 ± 1.2	70.3 ± 0.9	79.0 ± 0.4	54.2 ± 7.3	61.1 ± 7.0	59.6 ± 4.5	67.6 ± 2.4	54.9 ± 1.9
GAT [208]	83.0 ± 0.7	72.5 ± 0.7	79.0 ± 0.3	56.3 ± 4.3	57.9 ± 6.1	57.8 ± 4.3	65.0 ± 3.7	51.3 ± 2.5
NLMLP [129]	68.5 ± 1.9	61.2 ± 1.6	71.8 ± 0.9	84.9 ± 5.7	85.4 ± 3.8	87.3 ± 4.3	50.7 ± 2.2	33.7 ± 1.5
NLGCN [129]	79.4 ± 1.5	75.2 ± 1.4	77.9 ± 0.7	57.6 ± 5.5	65.5 ± 6.6	60.2 ± 5.3	70.1 ± 2.9	59.0 ± 1.2
NLGAT [129]	80.1 ± 1.3	77.2 ± 1.5	78.1 ± 0.7	54.7 ± 7.6	62.6 ± 7.1	56.9 ± 7.3	65.7 ± 1.4	56.8 ± 2.5
Geom-GCN-I [160]	80.0 ± 1.2	77.3 ± 0.8	78.2 ± 0.5	56.7 ± 8.6	57.5 ± 5.8	58.2 ± 4.9	60.3 ± 2.7	33.3 ± 1.4
PyGNN [52]	83.3 ± 0.9	72.9 ± 0.8	79.8 ± 0.4	75.3 ± 8.6	79.2 ± 4.6	76.9 ± 4.5	65.4 ± 2.5	59.3 ± 1.7
UFGCONV-S [253]	83.0 ± 0.5	71.0 ± 0.6	79.4 ± 0.4	67.8 ± 8.0	75.9 ± 4.8	72.4 ± 4.2	62.8 ± 2.3	57.6 ± 1.5
UFGCONV-R [253]	83.6 ± 0.6	72.7 ± 0.6	79.9 ± 0.1	68.9 ± 8.3	77.2 ± 4.7	73.5 ± 4.1	63.1 ± 2.4	57.2 ± 1.5
MM-FGCN (Ours)	$\mathbf{84.4}^* \pm 0.5$	$\mathbf{73.9}^* \pm 0.6$	$\mathbf{80.7}^* \pm 0.2$	$\mathbf{88.9}^* \pm 8.3$	$\mathbf{86.1}^* \pm 4.5$	$\mathbf{88.5}^* \pm 4.1$	$\mathbf{73.97}^* \pm 2.1$	$\mathbf{67.5}^* \pm 1.2$

Table 4.4: Performance comparison for graph property prediction. QM7 is a regression task in MSE; others are for classification in test accuracy in percentage. The results with the best performance are highlighted with *****.

Pooling Operators	PROTEINS (↑)	Mutagenicity (↑)	D&D (↑)	NCI1 (↑)	Ogbg-molhiv (↑)	QM7 (↓)
TOPKPool	73.48 ± 3.57	79.84 ± 2.46	74.87 ± 4.12	75.11 ± 3.45	78.14 ± 0.62	175.41 ± 3.16
AttentionPool	73.93 ± 5.37	80.25 ± 2.22	77.48 ± 2.65	74.04 ± 1.27	74.44 ± 2.12	177.99 ± 2.22
SAGPool	75.89 ± 2.91	79.86 ± 2.36	74.96 ± 3.60	76.30 ± 1.53	75.26 ± 2.29	41.93 ± 1.14
SUMPool	74.91 ± 4.08	80.69 ± 3.26	78.91 ± 3.37	76.96 ± 1.70	77.41 ± 1.16	42.09 ± 0.91
MAX	73.57 ± 3.94	78.83 ± 1.70	75.80 ± 4.11	75.96 ± 1.82	78.16 ± 1.33	177.48 ± 4.70
MEAN	73.13 ± 3.18	80.37 ± 2.44	76.89 ± 2.23	73.70 ± 2.55	78.21 ± 0.90	177.49 ± 4.69
UFGPool-SUM	77.77 ± 2.60	81.59 ± 1.40	80.92 ± 1.68	77.88 ± 1.24	78.80 ± 0.56	41.74 ± 0.84
UFGPool-SPECTRUM	77.23 ± 2.40	82.05 ± 1.28	79.83 ± 1.88	78.36 ± 0.77	78.36 ± 0.77	41.67 ± 0.95
MM-FGPool (Ours)	$\mathbf{78.07}^* \pm 2.36$	$\mathbf{83.91}^* \pm 1.32$	$\mathbf{81.51}^* \pm 1.55$	$\mathbf{78.57}^* \pm 0.82$	$\mathbf{79.12}^* \pm 0.85$	$\mathbf{41.19}^* \pm 0.88$

where the node homophily is diminished and conventional models based on low-pass filters such as GCN struggle to capture effective graph representations. In contrast, MM-FGCN demonstrates its capability of learning a multiresolution framelet transform that dynamically adapts to the characteristics of each graph dataset.

4.4.2 Graph Classification

We assess the efficacy of our MM-FGCN on six benchmark graph classification and regression datasets, including D&D [36], PROTEINS [36], NCI1 [214], Mutagenicity [94], Ogbg-molhiv [82], and QM7 [9]. Following the configuration of [253], each dataset is

Table 4.5: Ablation study on the meta-framelet learner and the meta-learning algorithm. Test accuracy (in percentage) with standard deviation after ±. are reported.

Methods	Graph Classification			Node Classification					
	Mutagenicity	D&D	NCI1	Cora	Citeseer	Cornell	Texas	Chameleon	Squirrel
(a) Haar-type	81.4 ± 1.4	80.9 ± 1.7	75.8 ± 1.3	83.3 ± 0.5	72.7 ± 0.7	77.8 ± 7.9	75.6 ± 9.8	54.6 ± 6.6	52.2 ± 2.1
(b) Linear-type	81.6 ± 1.4	80.6 ± 1.8	75.1 ± 1.1	83.0 ± 0.6	71.8 ± 0.9	76.1 ± 7.5	72.8 ± 9.5	54.3 ± 2.1	54.7 ± 1.7
(c) Quadratic-type	81.1 ± 1.3	80.3 ± 1.9	74.8 ± 1.4	82.7 ± 0.7	71.1 ± 0.7	76.7 ± 8.9	72.2 ± 9.4	57.5 ± 2.7	53.1 ± 1.8
(d) Trainable framelet transforms	82.3 ± 1.4	81.0 ± 1.7	75.9 ± 0.9	82.9 ± 0.5	72.2 ± 0.7	78.2 ± 8.5	77.9 ± 9.2	62.6 ± 2.6	59.2 ± 2.2
(e) **MM-FGCN (Ours)**	$\mathbf{83.9}^* \pm 1.3$	$\mathbf{81.5}^* \pm 1.5$	$\mathbf{78.5}^* \pm 0.8$	$\mathbf{84.4}^* \pm 0.5$	$\mathbf{73.9}^* \pm 0.6$	$\mathbf{88.9}^* \pm 8.3$	$\mathbf{86.1}^* \pm 4.5$	$\mathbf{73.9}^* \pm 2.1$	$\mathbf{67.5}^* \pm 1.2$

split into a training, validation, and test set by a ratio of 80%, 10%, and 10%. The results are averaged over 10 independent repetitions. We also compare our MM-FGPool with graph classification methods based on the conventional GCN backbone together with various state-of-the-art pooling strategies, including SUM, MEAN, MAX pooling, TOPKPool [46], AttentionPool [118], SAGPool [114], UFGPool-SUM, and UFGPool-SPECTRUM [253]. The results are shown in Table 4.4, and our model achieves the highest performance among all the baselines on the five datasets, demonstrating the effectiveness of MM-FGPool in aggregating graph information on various datasets.

4.4.3 Implementation details

We implement our model using PyTorch. We set the default number of filters as four, which is suitable for most of the datasets. The default Chebyshev approximation order is set to 6. The dimension of hidden variables is searched from $\{16, 32, 64\}$, and the level of filters are selected from $\{2, 3, 4, 5\}$. Other hyperparameters are set at: 0.001 for the learning rate, 0.001 for weight decay, 0.5 for dropout, and 2 for the number of MM-FGConv layers. These hyper-parameters are used for both node and graph classification tasks. For the graph classification task, we further apply our proposed MM-FGPool operation as elaborated in Section 4.3, followed by a linear classifier. For baseline methods in node classification, we adopt the code from the author's released implementation with the default settings. For graph classification, we adopt the experiment setting form [253], where we use two-layer GCN networks followed by the pooling methods listed in Table 4.4. Specifically, this experiment setting is comparable to our model's design, where we also adopt two convolutional layers and one pooling layer.

4.4.4 Ablation Studies

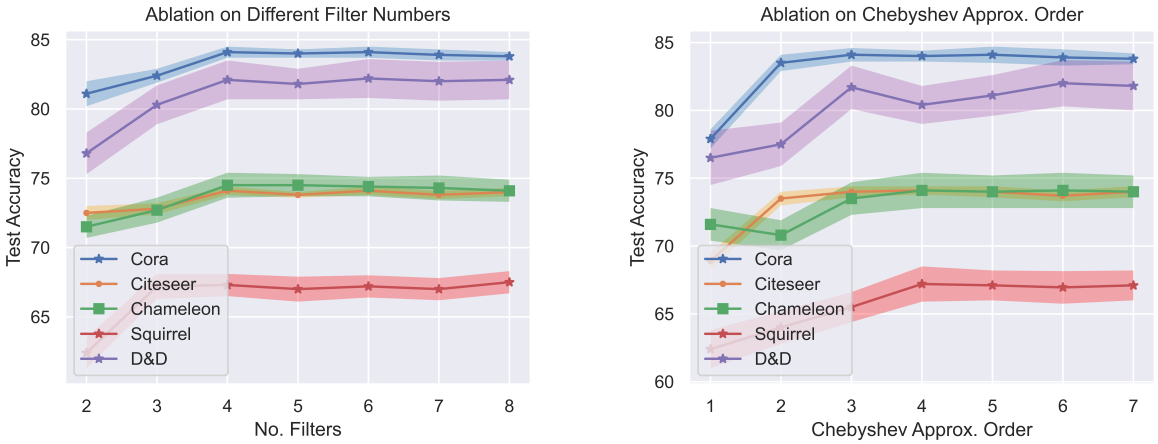
To validate the benefits of using a meta-framelet learner, in Table 4.5, we show the MM-FGCN variants with handcrafted filters [39] (e.g. (a) Haar-type, (b) linear-type, (c) quadratic-type framelet filters). To assess the performance improvement achieved by the meta-learning algorithm elaborated in Algorithm 4, we compare it against a direct training scheme where both $\boldsymbol{\theta}$ and $\boldsymbol{\xi}$ are updated simultaneously, as shown in row (d) trainable framelet transforms of Table 4.5. According to the results, models with trainable meta-framelet generators outperform those with fixed and handcrafted graph transforms, highlighting the necessity of using trainable graph transforms for enhanced performance. Furthermore, using a meta-framelet learner indeed brings performance gain compared to using directly trainable filters, showing that the meta-framelet learner enhances the capacity of MM-FGCN. We also show that meta-learning contributes to improvement in the generalization performance of our MM-FGCN, leading to more discriminative graph representations.

Ablation on the number of the meta-framelet generators. We analyze how the size of the meta-framelet generator (i.e. a set of spectral filters), I, affects the performance of MM-FGCN. With an insufficient amount of meta-filters, the model may fail to learn the optimal frequency partition and cannot disentangle graph signals into desirable frequency components. Intuitively, an overly small I may hinder the learning of discriminative graph representations. Conversely, a large I improves the precision of frequency partition learning but also increases computational expenses. This requires us to strike a balance between the by selectively choosing the value of I.

We evaluate the performance of MM-FGCN with different choices of I over Cora, Citeseer, Chameleon, Squirrel, and D&D datasets. As shown in Figure 4.4 (a), the Meta-FCGN is able to achieve state-of-the-art performance by constructing the meta-framelet generator with only 3 filters. Overall, the model performance is stable and robust across different choices of I. In general, we recommend setting $I = 4$ for effective and efficient implementation.

Ablation on the order of Chebyshev approximation. Recall that the Chebyshev approximation trick [32] is applied in Algorithm 3 for efficient computation of each $g_{r,i}^{\boldsymbol{\omega}}(\mathbf{L})$. Broadly speaking, using a higher-order Chebyshev approximation leads to a

smaller approximation error w.r.t the meta-generator, but creates a greater computational overhead. As illustrated in Figure 4.4 (b), the performance of MM-FGCN is robust to Chebyshev approximation of different orders. Empirically, the MM-FGCN achieves optimal performance with a Chebyshev approximation of an order greater than 4. In contrast, a low-order Chebyshev approximation order incurs an undesirable approximation error, which hinders graph representation learning and impairs model generalization. We recommend using a higher than 4-order Chebyshev approximation for good model performance.



(a) Ablation on the number of filters used in the meta-framelet generator.

(b) Ablation on the order of Chebyshev approximation.

Fig. 4.4: Ablation studies on MM-FGCN's hyperparameters.

4.4.5 Perturbation Resilience of MM-FGCN.

In this study, we add extra experiments to assess the perturbation resilience of our MM-FGCN against noise perturbation present in input graph data, which is ubiquitous in real-world datasets. Particularly, we train the MM-FGCN with corrupted data that are contaminated by random noise of various magnitudes. The noise magnitude is controlled by the noise ratios, which are defined as the amount of randomly deleted edges (or randomly flipped binary-valued features) divided by the number of untainted edges (or features). We then investigate how the performance of the resultant models varies when

the noise level change from 0 to 1. As illustrated in Figure 4.5, our MM-FGCN consistently outperforms the baselines with a remarkable margin even under the presence of considerable noise. Thus, the MM-FGCN demonstrates a strong noise resilience making it a highly promising solution for real-world applications.

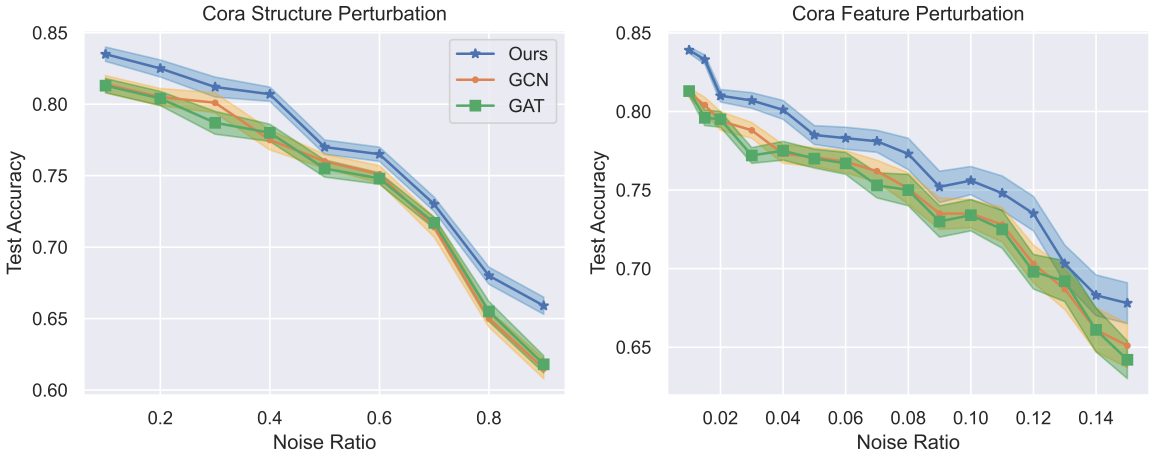


Fig. 4.5: Noise resilience experiments with the edge (left) and feature (right) noise perturbations on Cora.

4.4.6 Visualization of MM-FGCN Representation

In this section, we empirically show that the MM-FGCN is able to produce more discriminative graph representations than conventional GCN, on both assortative (e.g. Cora) and disassortative (e.g. Cornell) datasets. In order to assess the quality of the learned graph representation, we visualize the hidden features generated by the penultimate layer of both MM-FGCN and GCN via t-SNE [207]. As shown in Figure 4.6, the graph representation of our MM-FGCN is more spatially clustering than GCN, especially for the disassortative dataset that is more challenging for classification. In fact, the learned feature of MM-FGCN is strongly correlated to the label, which significantly facilitates node and graph classification tasks. In summary, the visualization of the learned hidden features validates that the adaptiveness and expressiveness of our MM-FGCN are beneficial to learning discriminative graph representations.

We also demonstrate the effectiveness of the MM-FGCN by visualizing the filters learned by MM-FGCN in Figure 4.6. We can observe that the filters learned from the

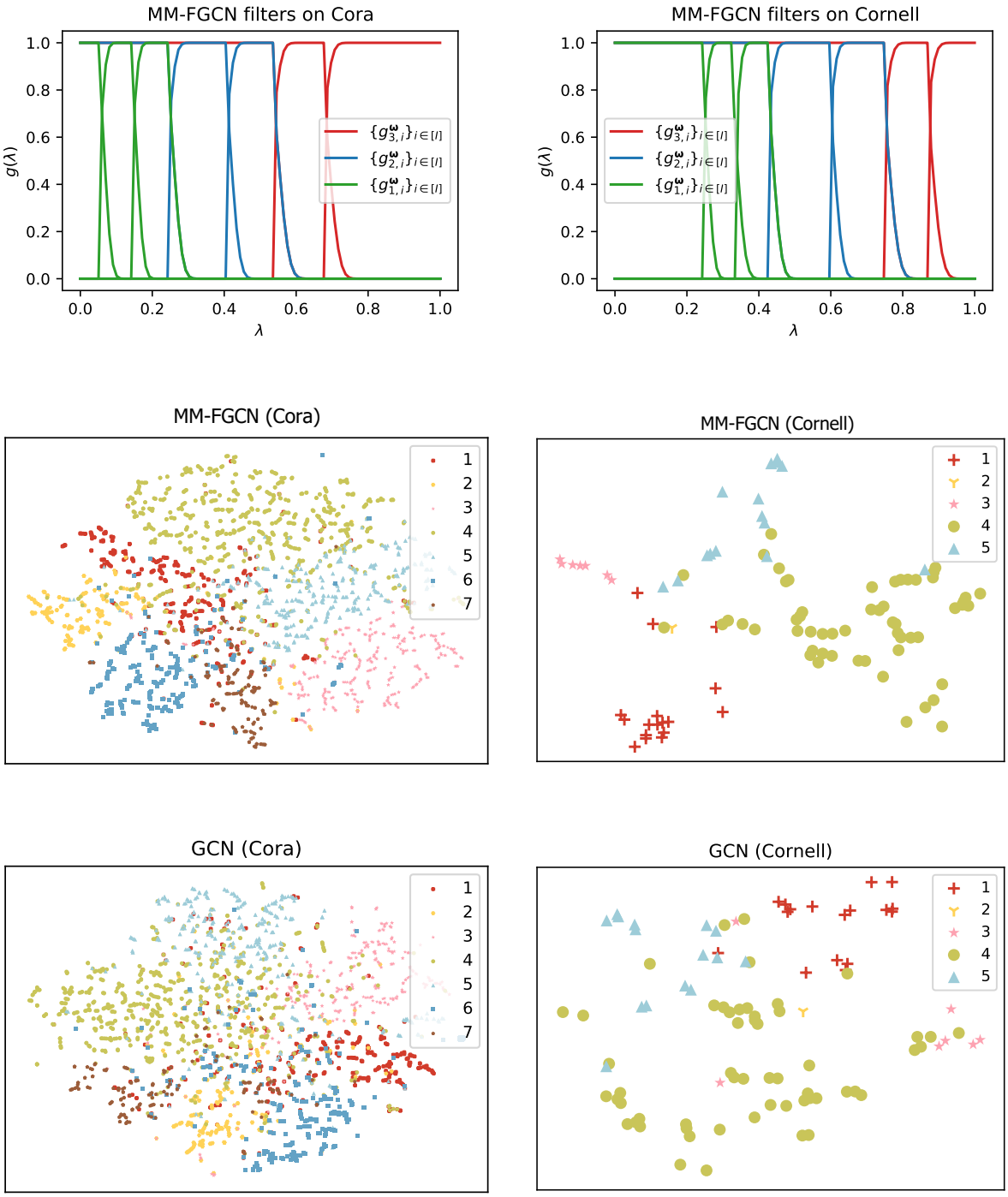


Fig. 4.6: Meta-framelet generator (row 1) and feature visualization (row 2-3) on the test and validation sets of Cora (left, assortative) and Cornell (right, disassortative) datasets using MM-FGCN and GCN.

Cora dataset (assortative) more concentrate on the low-frequency signals than the filters learned from the Cornell dataset (disassortative). Due to the assortative properties, aggregating local information with low-frequency signals can benefit the model's performance on the Cora dataset. In contrast, as elaborated in [10], high-frequency signals are useful for disassortative networks, which corresponds to our learned multi-resolution filters since more filters concentrate on the high-frequency part, providing a comprehensive feature extraction on the high-frequency signal. This phenomenon shows the adaptivity of our MM-FGCN on different types of graphs.

4.5 Proof Details

Proof: [Proof of Proposition 4.1] According to the discrete tight framelet transform theory (Theorem 2.1 and Theorem 3.1 in Dong2017), the series of progressive resolution subspaces $\{V_r\}_r$ with $V_r = \text{span}\left(\{\boldsymbol{\varphi}_{riv}\}_{i,v}\right)$ inherently satisfies the denseness, translation, and dilation properties, making it a set of desirable multiresolution bases for graph domain data. To complete the proof, one only need to verify the tightness of the MMFS, i.e. $\boldsymbol{\Phi}_{\text{MM}}\boldsymbol{\Phi}_{\text{MM}}^{\top}\mathbf{x} = \mathbf{x}$ holds for any graph signal $\mathbf{x}$. Let $\mathcal{I} = ([R]\times[I])\backslash\{(r,1) : 1 \leqslant r < R\}$,

we have

$$\boldsymbol{\Phi}_{\mathrm{MM}}\boldsymbol{\Phi}_{\mathrm{MM}}^{\top} = \sum_{r,i,v} \boldsymbol{\varphi}_{riv}\boldsymbol{\varphi}_{riv} \tag{4.10}$$

$$= \sum_{(r,i)\in\mathcal{I}} \left(\mathbf{U} g^{\boldsymbol{\omega}}_{r,i}(\gamma^{-J+r}\mathbf{L})\mathbf{U}^{\top}\right)\left(\mathbf{U} g^{\boldsymbol{\omega}}_{r,i}(\gamma^{-J+r}\mathbf{L})\mathbf{U}^{\top}\right)^{\top} \tag{4.11}$$

$$= \mathbf{U}\left(\sum_{(r,i)\in\mathcal{I}} {g^{\boldsymbol{\omega}}_{r,i}}^{2}(\gamma^{-J+r}\mathbf{L})\right)\mathbf{U}^{\top} \tag{4.12}$$

$$= \mathbf{U}\Bigg(\Bigg(\sum_{1<i\leqslant I, 1\leqslant r<R} {g^{\boldsymbol{\omega}}_{1,i}}^{2}(\gamma^{-J+1}\mathbf{L})\Bigg) + \Bigg(\sum_{i\in[I]} {g^{\boldsymbol{\omega}}_{R,i}}^{2}(\gamma^{-J+R}\mathbf{L}){g^{\boldsymbol{\omega}}_{1,1}}^{2}(\gamma^{-J+R-1}\mathbf{L})\cdots{g^{\boldsymbol{\omega}}_{1,1}}^{2}(\gamma^{-J+1}\mathbf{L})\Bigg)\Bigg)\mathbf{U}^{\top} \tag{4.13}$$

$$= \mathbf{U}\Bigg(\Bigg(\sum_{1<i\leqslant I, 1\leqslant r<R} {g^{\boldsymbol{\omega}}_{1,i}}^{2}(\gamma^{-J+1}\mathbf{L})\Bigg) + \Bigg({g^{\boldsymbol{\omega}}_{1,1}}^{2}(\gamma^{-J+R-1}\mathbf{L})\cdots{g^{\boldsymbol{\omega}}_{1,1}}^{2}(\gamma^{-J+1}\mathbf{L})\Bigg)\Bigg)\mathbf{U}^{\top} \tag{4.14}$$

$$= \mathbf{U}\Bigg(\Bigg(\sum_{1<i\leqslant I, 1\leqslant r<R} {g^{\boldsymbol{\omega}}_{1,i}}^{2}(\gamma^{-J+1}\mathbf{L})\Bigg) + g^{\boldsymbol{\omega}}_{R-1,1}(\gamma^{-J+R-1}\mathbf{L})\Bigg)\mathbf{U}^{\top} \tag{4.15}$$

$$= \mathbf{U}\Bigg(\Bigg(\sum_{1<i\leqslant I, 1\leqslant r<R-1} {g^{\boldsymbol{\omega}}_{1,i}}^{2}(\gamma^{-J+1}\mathbf{L})\Bigg) + g^{\boldsymbol{\omega}}_{R-2,1}(\gamma^{-J+R-2}\mathbf{L})\Bigg)\mathbf{U}^{\top} \tag{4.16}$$

$$\vdots \tag{4.17}$$

$$= \mathbf{U}\Bigg(\sum_{1\leqslant i\leqslant I} {g^{\boldsymbol{\omega}}_{1,i}}^{2}(\gamma^{-J+1}\mathbf{L})\Bigg)\mathbf{U}^{\top} = \mathbf{I}, \tag{4.18}$$

which completes the proof.

4.6 Conclusion

In this Chapter, we present MM-FGCN, a spectral-based model for adaptive multiresolution representation learning for varying graph instances. Our MM-FGCN model is equipped with a set of trainable multiresolution bases, which can be simply and efficiently constructed based on a set of meta-band-pass filters. By optimizing the meta-filters, MM-FGCN learns an adaptive frequency partition of the graph spectrum domain, enabling us to perform a customized multiresolution transform on each graph instance.

Comprehensive experiments show that our proposed method exhibits high performance and adaptivity to various types of graphs, including graph and node classification for dissortative and assortative graphs from various domains.

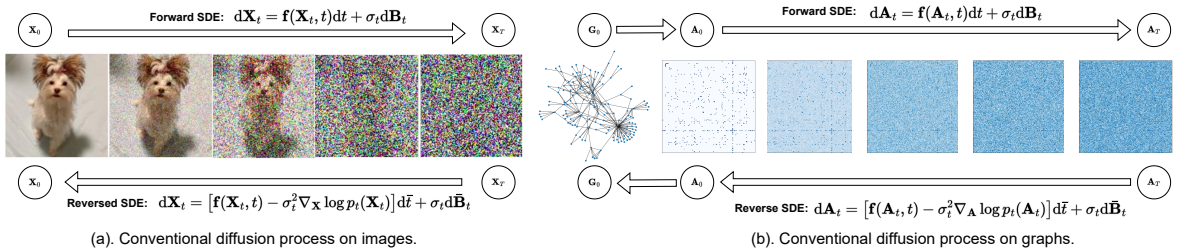
Chapter 5

Fast Graph Generation via Spectral Diffusion

5.1 Overview

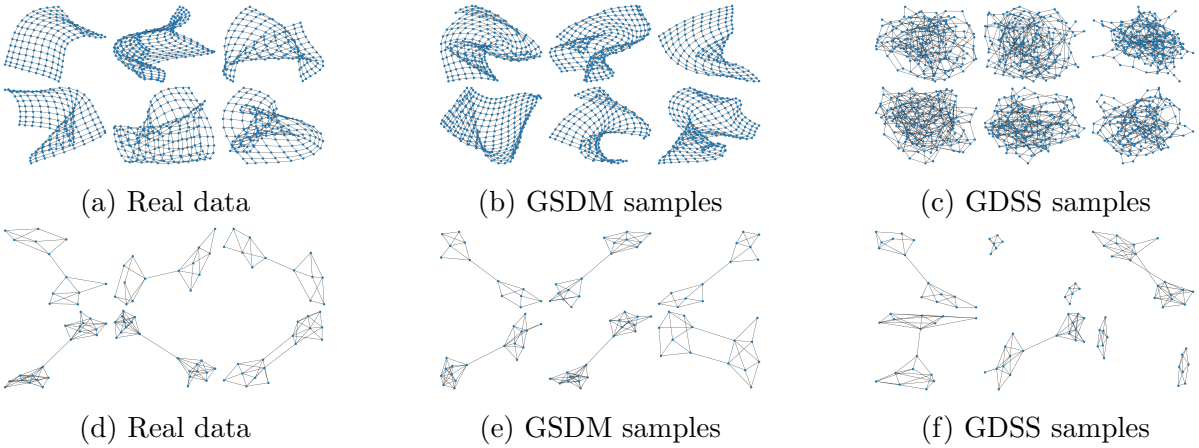
One of the primary objectives of utilizing graph feature extraction is to acquire the ability to create suitable graphs. In this chapter, our emphasis is on introducing a unique approach to generating graphs. Learning to generate graph-structural data not only requires knowing the nodes' feature distribution but also a deep understanding of the underlying graph topology, which is essential to modeling various graph instances, such as social networks [215,236], molecule structures [183,245], neural architectures [113], recommender systems [126], etc. Conventional likelihood-based graph generative models, e.g. GraphGAN [216], GraphVAE [186] and GraphRNN [238], have demonstrated great strength on graph generation tasks. In general, a likelihood-based model is designed to learn the likelihood function of the underlying graph data distribution, with which one can draw new samples with preserved graph properties from the distribution of interest. However, most likelihood-based generative models suffer from either limited quality of modeling graph structures, or considerable computational burden [91].

Recently, a series of diffusion-based generative models have been proposed to overcome the limitations of likelihood-based models. Although being originally established for image generation [187], diffusion models exhibit great success in graph generation tasks with complex graph structural properties [91, 153]. Roughly speaking, diffusion is a mathematical technique that involves a Stochastic Differential Equation (SDE) used to smoothly transform real data into pure noise by adding more noise. Diffusion models are a class of probabilistic generative models that use this technique in reverse to generate representative data samples from noise. These models are trained to learn the underlying structure of a dataset by simulating how data points gradually become blurry due to added noise. Therefore, a well-trained diffusion model allows us to restore the original data by reversing the diffusion process and gradually removing the noise from the blurred sample. The first graph diffusion model through SDEs, coined Graph Diffusion via SDE Systems (GDSS) [91], is designed to simultaneously generate node features and adjacency matrix via reversed diffusion. Similar to image diffusion models [187,188], at each diffusion step, GDSS directly inserts standard Gaussian noise to both node features and the adjacency matrix. Meanwhile, two separate neural networks are trained to learn the score functions of the node features and adjacency matrix, respectively.



(a). Conventional diffusion process on images. (b). Conventional diffusion process on graphs.

Fig. 5.1: Illustration of the difference between applying conventional SDE diffusion on images (a) and graphs (b).



(a) Real data (b) GSDM samples (c) GDSS samples

(d) Real data (e) GSDM samples (f) GDSS samples

Fig. 5.2: Non-cherry-picked random samples from the testing set as well as samples generated by GSDM (ours) and GDSS [91], on Grid (top row) and Community-small (bottom row) datasets. For GDSS, we use the authors' released code and checkpoints to generate the samples.

However, unlike the densely distributed image data, graph adjacency matrices can be highly sparse, which makes isotropic Gaussian noise insertion incompatible with graph structural data. In these circumstances, as shown in Figure 5.1, there is a stark difference between the diffusion process on images and on graph adjacency matrices. As can be seen from the figure, the image corrupted by full-rank Gaussian noise exhibits recognizable numerical patterns along the early- and middle-stage of forward diffusion. However, the corrupted sparse graph adjacency matrix degenerates into a dense matrix with uniformly distributed entries in a few diffusion steps. In intuition, Figure 5.1 implies that standard diffusion SDEs with full-rank isotropic noise insertion are destructive of learning graph topology and feature representations. Theoretically speaking, for extremely sparse graphs (e.g. social networks, molecules) with low-rank adjacency matrices, the adjacency score

functions are supported on a low-dimensional manifold embedded in the full adjacency matrix space. Thus, directly applying diffusion models on graph topology generation is not desirable: once the diffusion SDE is run in the full space of the adjacency matrix, lethal noise will be injected into the out-of-support regions and drives the signal-to-noise ratio to be essentially zero, which is fatal for training score networks.

Even for densely connected graphs, the standard diffusion model is problematic for topology generation. Unlike image pixels that are merely locally correlated, an adjacency matrix governs the message-passing pattern of the whole graph. Thus, isotropic Gaussian noise insertion severely distorts the message-passing pattern, by blindly encouraging message passing on sparsely connected parts, which impedes the representation learning of sparse regions.

In order to establish a graph-friendly diffusion model, one should design an appropriate diffusion scheme that is compatible with the graph topology structure. To this end, we propose the Graph Spectral Diffusion Model (GSDM), which is driven by diffusion SDEs on both the node feature space and the graph spectrum space. At each diffusion step, instead of corrupting the entire adjacency matrix, our method confines the Gaussian insertion to the graph spectrum space, i.e. the eigenvalue matrix of the adjacency matrix. This novel diffusion scheme enables us to perform smooth transformations on graph data during both the training and sampling phases. As illustrated in Figure 5.2, our proposed GSDM significantly outperforms the standard graph diffusion model (GDSS [91]) in terms of graph generation quality and plausibility. For the Grid dataset shown in the top row of the figure, GDSS samples seem to be merely chaotic clusters, while GSDM samples exhibit smooth surface-like patterns that are visually similar to real data. For the Community-small dataset shown in the bottom row of the figure, GDSS fails to capture the link between two communities on some samples, while GSDM is able to capture dumbbell-like patterns (two clusters connected by one edge) as well as butterfly-like patterns (two clusters connected by two edges). This implies that GDSS's generation not only fails to mimic the observed topology distribution but also suffers from capturing challenging details of the data such as links between communities. In contrast, GSDM is capable of generating high-quality graphs that are topologically similar to the real data, while retaining critical details.

We empirically evaluate the capability of our proposed GSDM on generic graph generation tasks, by evaluating the generation quality on both synthetic and real-world graph datasets. As shown in Section 5.5, GSDM outperforms existing one-shot generative models on various datasets, while achieving competitive performance to autoregressive models. Further molecule generation experiments show that our GSDM outperforms the state-of-the-art baselines, demonstrating that our proposed spectral diffusion model is capable of capturing complicated dependency between nodes and edges. Our main contributions are 3 folds:

- We propose a novel Graph Spectral Diffusion Model (GSDM) for fast and quality graph generation. Our method overcomes the limitations of existing graph diffusion models by leveraging diffusion SDEs on both the node feature and graph spectrum spaces.
- Through the lens of stochastic analysis, we prove that GSDM enjoys a substantially stronger performance guarantee than the standard graph diffusion model. Our proposed spectral diffusion sharpens the reconstruction error bound from $\mathcal{O}(n^2 \exp(n^2))$ to $\mathcal{O}(n \exp(n))$, where n is the number of nodes.
- We evaluate GSDM on both synthetic and real-world graph generation tasks. GSDM outperforms all existing graph generative models and also achieves evidently higher computational efficiency compared to existing graph diffusion models.

5.2 Preliminaries

In this chapter, we denote the probability space of interest as $(\Omega, \mathcal{F}, \mathbb{P})$ and $(\mathcal{F}_t)_{t\in\mathbb{R}}$ be a filtration, i.e. a sequence of increasing sub-σ-algebra of $\mathcal{F}$. Without specification, we denote $(\mathbf{B}_t)_{t\in\mathbb{R}}$ as the d-dimensional standard Brownian motion on the filtered probabilistic space $(\Omega, \mathcal{F}, \mathbb{P}, (\mathcal{F}_t)_{t\in\mathbb{R}})$. The distribution, support set, and expectation of a random variable $\mathbf{z}$ are defined as $\text{law}(\mathbf{z})$, $\text{supp}(\mathbf{z})$ and $\mathbb{E}[\mathbf{z}]$. $\mathbf{y}|\mathbf{z}$ denotes the distribution of $\mathbf{y}$ conditioned on $\mathbf{z}$. $\text{Unif}(A)$ denotes the uniform distribution on a set A. $\|\cdot\|$ denotes the standard Euclid norm. $\|\cdot\|_\infty$ and $\|\cdot\|_{\text{lip}}$ denotes the supremum norm and Lipschitz norm of a function. $[\cdot]$ denotes the flooring function.

5.2.1 Score-based Generative Diffusion Model

A generative model refers to a mapping $g_{\boldsymbol{\theta}} : \mathbb{R}^d \mapsto \mathbb{R}^d$, which maps a simple known priori π to a complicated data distribution $\mathcal{D}$. Once the model $g_{\boldsymbol{\theta}}$ is sufficiently trained on N i.i.d samples from $\mathcal{D}$, denoted by $\mathcal{S}$, it enables us to generate plausible instances from $\mathcal{D}$ directly, by sampling from $g_{\boldsymbol{\theta}}(\boldsymbol{\varepsilon})$, $\boldsymbol{\varepsilon} \sim \pi$. Unlike conventional generative models, e.g. VAE and GAN, which treat $\mathcal{D}$ as a unilateral transformation of π, diffusion models consider the bilateral relation between $\mathcal{D}$ and π from the perspective of SDE. Given an SDE traveling from $\mathcal{D}$ to π, the corresponding reversed time SDE enables us to backtrack from noisy priori to the distribution of interest.

Lemma 5.1 (Forward Diffusion and Reversed Time SDE [3])**.**
The Forward Diffusion refers to the following SDE

$$\mathbf{z}_0 \sim \mathcal{D},\ \mathrm{d}\mathbf{z}_t = \mathbf{f}(\mathbf{z}_t, t)\mathrm{d}t + \sigma_t \mathrm{d}\mathbf{B}_t,\ t \in [0, 1], \tag{5.1}$$

where $\mathbf{f}(\cdot, t) : \mathbb{R}^d \mapsto \mathbb{R}^d$ is the drift function, $\sigma_t : [0, 1] \mapsto \mathbb{R}$ be a scalar diffusion function. Let $p_t(\cdot)$ be the probability density function of $\mathbf{z}_t$, then the Reversed Time SDE is given by

$$\mathrm{d}\bar{\mathbf{z}}_t = (\mathbf{f}(\bar{\mathbf{z}}_t, t) - \sigma_t^2 \nabla \log p_t(\bar{\mathbf{z}}_t))\mathrm{d}\bar{t} + \sigma_t \mathrm{d}\bar{\mathbf{B}}_t, \tag{5.2}$$

where $\bar{\mathbf{z}}_1 \sim \mathbf{z}_1$, $t \in [0, 1]$, $\mathrm{d}\bar{t} = -\mathrm{d}t$ is the negative infinitesimal time step, $(\bar{\mathbf{B}}_t)_{t\in\mathbb{R}}$ is a reversed time Brownian motion w.r.t $(\Omega, \mathcal{F}, \mathbb{P}, (\bar{\mathcal{F}}_t)_{t\in\mathbb{R}})$, and $(\bar{\mathcal{F}}_t)_{t\in\mathbb{R}}$ is the corresponding decreasing filtration, and $\nabla \log p_t(\cdot)$ is the score function.

During the forward diffusion process, with a carefully designed $\mathbf{f}(\cdot, t)$, the original data is perturbed by Gaussian noise with increasing magnitude, and it is assumed to be gradually corrupted to a truly noisy signal (priori), i.e. $\text{law}(\mathbf{z}_1) = \pi$. In order to draw new data from $\mathcal{D}$ via the reversed time SDE, one needs to learn the unknown score function $\nabla \log p_t(\cdot)$ with a neural network $s_{\boldsymbol{\theta}}(\cdot) : \mathbb{R}^d \mapsto \mathbb{R}^d$, by minimizing the following explicit score matching error:

$$\mathcal{E}(\boldsymbol{\theta}) \triangleq \mathbb{E}_{\mathbf{z}\sim\mathcal{D}}\mathbb{E}_{\mathbf{z}_t|\mathbf{z}}\|s_{\boldsymbol{\theta}}(\mathbf{z}_t) - \nabla \log p_t(\mathbf{z}_t)\|^2. \tag{5.3}$$

In practice, we employ a Gaussian priori π. For each sample $\mathbf{z}^i \in \mathcal{S}$, we first generate a sequence of corrupted data $\{\mathbf{z}_{t_j}^i\}_{j=1}^T$ by discretizing (5.1). To learn the score network $s_{\boldsymbol{\theta}}(\cdot)$, one can minimize a more tractable denoising score matching objective $\hat{\mathcal{E}}(\boldsymbol{\theta})$

$$\hat{\mathcal{E}}(\boldsymbol{\theta}) \triangleq \mathbb{E}_{\mathbf{z}\sim\text{Unif}(\mathcal{S})}\mathbb{E}_{\mathbf{z}_t|\mathbf{z}}\|s_{\boldsymbol{\theta}}(\mathbf{z}_t) - \nabla \log p(\mathbf{z}_t|\mathbf{z})\|^2, \tag{5.4}$$

which has been proven to be equivalent to $\mathcal{E}(\boldsymbol{\theta})$ in [213]. Given a well trained score network $s_{\boldsymbol{\theta}^*}(\cdot)$, one is able to generate plausible data from π via solving the learned reversed-time SDE

$$\mathrm{d}\hat{\mathbf{z}}_t = (\mathbf{f}(\hat{\mathbf{z}}_t, t) - \sigma_t^2 s_{\boldsymbol{\theta}^*}(\hat{\mathbf{z}}_t))\mathrm{d}\bar{t} + \sigma_t \mathrm{d}\bar{\mathbf{B}}_t, \tag{5.5}$$

where $\hat{\mathbf{z}}_1 \sim \pi$ and $t \in [0, 1]$. Ideally, the learned reversed time SDE should lead us towards $\mathcal{D}$, i.e. $\text{law}(\hat{\mathbf{z}}_0) = \mathcal{D}$.

5.3 Methodology

In this section, we establish the Graph Spectral Diffusion Model (GSDM) for fast and effective graph data generation. In Section 4.1, we briefly review the standard score-based graph diffusion model [91]. In Section 4.2, we formally introduce our GSDM algorithm and its α-quantile variants. In Section 4.3, we provide theoretical analyses to justify the efficacy of GSDM on graph data generation.

5.3.1 Standard Graph Diffusion Model

A graph with n nodes is defined as $\mathbf{G} \triangleq (\mathbf{X}, \mathbf{A}) \in \mathbb{R}^{n\times d} \times \mathbb{R}^{n\times n}$, where $\mathbf{X} \in \mathbb{R}^{n\times d}$ is the node feature matrix with dimension d and $\mathbf{A} \in \mathbb{R}^{n\times n}$ denotes the adjacency matrix. A graph generative model aims to learn the underlying data distribution, say $\mathbf{G} \sim \mathcal{G}$, which is a joint distribution of both $\mathbf{X}$ and $\mathbf{A}$. Note that, if $(\mathbf{X}, \mathbf{A})$ is treated as a whole and omits the intrinsic graph structure, the aforementioned score-based generation framework can be parallelly extended to the graph generation setting, which yields the standard graph diffusion model, i.e. GDSS [91]. Roughly speaking, for each graph sample $(\mathbf{X}, \mathbf{A})$, we first generate a sequence of perturbed graphs $\{(\mathbf{X}_{t_i}, \mathbf{A}_{t_i})\}_{i=1}^{T}$ via forward diffusion. Then, we train two score networks $s_{\boldsymbol{\theta}}(\cdot)$ and $s_{\boldsymbol{\phi}}(\cdot)$ to learn the score functions for both $\mathbf{X}_t$ and $\mathbf{A}_t$, with which we can generate new data from π by running the reversed time SDE.

Definition 5.2 (Graph Diffusion SDEs)**.**
The Forward Graph Diffusion refers to the following SDE system

$$\begin{cases} \mathrm{d}\mathbf{X}_t = \mathbf{f}^X(\mathbf{X}_t, t)\mathrm{d}t + \sigma_{X,t}\mathrm{d}\mathbf{B}_t^X, \\ \mathrm{d}\mathbf{A}_t = \mathbf{f}^A(\mathbf{A}_t, t)\mathrm{d}t + \sigma_{A,t}\mathrm{d}\mathbf{B}_t^A, \end{cases} \tag{5.6}$$

where $(\mathbf{X}_0, \mathbf{A}_0) \sim \mathcal{G}$, $t \in [0, 1]$, $\mathbf{f}^X(\cdot, t) : \mathbb{R}^{n\times d} \mapsto \mathbb{R}^{n\times d}$ *and* $\mathbf{f}^A(\cdot, t) : \mathbb{R}^{n\times n} \mapsto \mathbb{R}^{n\times n}$ *is the drift functions for nodes feature and adjacency matrix;* $\sigma_{X,t}, \sigma_{A,t}$ *are the scalar diffusion terms (a.k.a noise schedule function), and* $(\mathbf{B}_t^X)_{t\in\mathbb{R}}$ *and* $(\mathbf{B}_t^A)_{t\in\mathbb{R}}$ *are standard Brownian motions on* $\mathbb{R}^{n\times d}$ *and* $\mathbb{R}^{n\times n}$*, respectively.*

To alleviate the computational burden of calculating the drift w.r.t the high dimensional $\mathbf{G}$, the drift term of forward graph diffusion is disentangled into $\mathbf{f}^X(\cdot, t)$ and $\mathbf{f}^A(\cdot, t)$. Again, Lemma 5.1 guarantees the existence of the reversed time SDEs for graph diffusion.

Corollary 5.1 (Reversed Time SDEs for Graph Diffusion)**.** *The reversed time SDE system of* (5.6) *is given by*

$$\begin{cases} \mathrm{d}\bar{\mathbf{X}}_t = \left(\mathbf{f}^X(\bar{\mathbf{X}}_t, t) - \sigma_{X,t}^2 \nabla_{\mathbf{X}} \log p_t(\bar{\mathbf{X}}_t, \bar{\mathbf{A}}_t)\right)\mathrm{d}\bar{t} + \sigma_{X,t}\mathrm{d}\bar{\mathbf{B}}_t^X, \\ \mathrm{d}\bar{\mathbf{A}}_t = \left(\mathbf{f}^A(\bar{\mathbf{A}}_t, t) - \sigma_{A,t}^2 \nabla_{\mathbf{A}} \log p_t(\bar{\mathbf{X}}_t, \bar{\mathbf{A}}_t)\right)\mathrm{d}\bar{t} + \sigma_{A,t}\mathrm{d}\bar{\mathbf{B}}_t^A, \end{cases} \tag{5.7}$$

where $(\bar{\mathbf{X}}_1, \bar{\mathbf{A}}_1) \sim \pi$, $t \in [0, 1]$, $\mathrm{d}\bar{t} = -\mathrm{d}t$ *is the negative infinitesimal time step,* $(\bar{\mathbf{B}}_t^X)_{t\in\mathbb{R}}$ *and* $(\bar{\mathbf{B}}_t^A)_{t\in\mathbb{R}}$ *are the reversed time standard Brownian motions induced by* (5.6).

The disentanglement of the drift functions implies the conditional independence $\mathbf{X}_t \perp \mathbf{A}_t|\mathbf{G}_0$, with which we can decompose $p_t(\bar{\mathbf{X}}_t, \bar{\mathbf{A}}_t)$ into $p_{t|0}(\mathbf{X}_t|\mathbf{X}_0) \cdot p_{t|0}(\mathbf{A}_t|\mathbf{A}_0)$, where $p_{t|0}(\cdot)$ denotes the density function of $\mathbf{G}_t|\mathbf{G_0}$. As proposed in [91], such conditional independence reduces the denoising score-matching objective to a simpler form

$$\widehat{\mathcal{E}}(\boldsymbol{\theta}) \triangleq \mathbb{E}_{\mathbf{G}\sim\mathrm{Unif}(\mathcal{S})}\mathbb{E}_{\mathbf{G}_t|\mathbf{G}}\|s_{\boldsymbol{\theta}}(\mathbf{G}_t) - \nabla \log p_{t|0}(\mathbf{X}_t|\mathbf{X}_0)\|^2, \tag{5.8}$$

$$\widehat{\mathcal{E}}(\boldsymbol{\phi}) \triangleq \mathbb{E}_{\mathbf{G}\sim\mathrm{Unif}(\mathcal{S})}\mathbb{E}_{\mathbf{G}_t|\mathbf{G}}\|s_{\boldsymbol{\phi}}(\mathbf{G}_t) - \nabla \log p_{t|0}(\mathbf{A}_t|\mathbf{A}_0)\|^2. \tag{5.9}$$

Hence, the training and sampling procedures can be directly borrowed from standard score-based models.

While GDSS is the first attempt at leveraging diffusion models on graph generation, its performance is hindered by the brute-force application of diffusion. For sparsely connected graphs, while the distribution of node features varies across datasets, the distribution of graph topology, i.e. adjacency matrix, resides in a low dimensional manifold. As mentioned in Section 1, an evident pattern of the adjacency matrices also implies that the true distribution of $\mathbf{A}$ is of low rank. In this case, the score-matching objective fails to provide consistency estimators. Although running a full rank diffusion on $\mathbf{A} \in \mathbb{R}^{n\times n}$ alleviates this issue by extending the support of corrupted data from the manifold to the full space, it inevitably introduces lethal noise to regions of zero probability density. As a consequence, the signal-to-noise ratio of regions out of $\mathrm{supp}(\mathbf{A})$ is essentially zero, which is a catastrophe for training the denoising score network. Note that such full-rank diffusion is also inappropriate for densely connected graph generation. This is because an isotropically corrupted adjacency matrix encourages delusive message passing on sparsely connected parts of the graph, which is destructive to the graph message passing pattern. Thus, standard diffusion can severely impair representation learning for sparse graph regions.

5.3.2 Graph Spectral Diffusion Model

To address these notorious yet ubiquitous issues, we propose a novel Graph Spectral Diffusion Model. For graph topology generation, in contrast to GDSS which is driven by a full-rank diffusion on the whole space $\mathbb{R}^{n\times n}$, our GSDM leverages low-rank diffusion SDEs on the n-dimensional spectrum manifold, e.g. the span of n-eigenvalues of $\mathbf{A}$. As we shall see later, GSDM achieves both robustness and computational efficiency, by exploiting the graph spectrum structure and running diffusion on an information-concentrated manifold.

Definition 5.3 (Graph Spectral Diffusion SDEs)**.**
Let the spectral decomposition of $\mathbf{A}$ *be* $\mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^\top$*, where columns of* $\mathbf{U}$ *are the orthonormal eigenvectors and* $\boldsymbol{\Lambda}$ *be the diagonal eigenvalue matrix, i.e. spectrum. The Forward Spectral Diffusion refers to the following SDE system*

$$\begin{cases} \mathrm{d}\mathbf{X}_t = & \mathbf{f}^X(\mathbf{X}_t, t)\mathrm{d}t + \sigma_{X,t}\mathrm{d}\mathbf{B}_t^X, \\ \mathrm{d}\boldsymbol{\Lambda}_t = & \mathbf{f}^\Lambda(\boldsymbol{\Lambda}_t, t)\mathrm{d}t + \sigma_{\Lambda,t}\mathrm{d}\mathbf{W}_t^\Lambda, \end{cases} \tag{5.10}$$

where $(\mathbf{X}_0, \mathbf{A}_0) \sim \mathcal{G}$, $\mathbf{A}_0 = \mathbf{U}_0\mathbf{\Lambda}_0\mathbf{U}_0^\top$, $t \in [0,1]$, $\mathbf{f}^X(\cdot,t) : \mathbb{R}^{n\times d} \mapsto \mathbb{R}^{n\times d}$ *is the drift for nodes feature;* $\mathbf{f}^\Lambda(\cdot,t) : \mathbb{R}^n \mapsto \mathbb{R}^n$ *is the drift for spectrum, which only acts on the diagonal entries;* $\sigma_{X,t}, \sigma_{\Lambda,t}$ *are the scalar diffusion terms (a.k.a noise schedule functions),* $(\mathbf{B}_t^X)_{t\in\mathbb{R}}$ *and* $(\mathbf{B}_t^\Lambda)_{t\in\mathbb{R}}$ *are standard Brownian motions on* $\mathbb{R}^{n\times d}$ *and* $\mathbb{R}^n$*, respectively; and* $\mathbf{W}_t^\Lambda \triangleq \mathrm{diag}(\mathbf{B}_t^\Lambda)$ *is a diagonal Brownian motion.*

As will be illustrated later, the evolution of $\mathbf{A}_t \triangleq \mathbf{U}_0\mathbf{\Lambda}_t\mathbf{U}_0^\top$ is driven by a n-dimensional Gaussian process. Hence, we prevent the corrupted adjacency matrix from rampaging around the full space. We are now ready to establish the reversed time spectral diffusion SDEs.

Corollary 5.2 (Reversed Time Spectral Diffusion SDEs)**.** *The reversed time Spectral Diffusion SDE system of* (5.10) *is given by*

$$\begin{cases} \mathrm{d}\bar{\mathbf{X}}_t = \Big(\mathbf{f}^X(\bar{\mathbf{X}}_t,t) - \sigma_{X,t}^2\nabla_{\mathbf{X}}\log p_t(\bar{\mathbf{X}}_t,\bar{\mathbf{\Lambda}}_t)\Big)\,\mathrm{d}\bar{t} + \sigma_{X,t}\mathrm{d}\bar{\mathbf{B}}_t^X, \\ \mathrm{d}\bar{\mathbf{\Lambda}}_t = \Big(\mathbf{f}^\Lambda(\bar{\mathbf{\Lambda}}_t,t) - \sigma_{\Lambda,t}^2\nabla_{\mathbf{\Lambda}}\log p_t(\bar{\mathbf{X}}_t,\bar{\mathbf{\Lambda}}_t)\Big)\,\mathrm{d}\bar{t} + \sigma_{\Lambda,t}\mathrm{d}\bar{\mathbf{W}}_t^\Lambda, \end{cases} \tag{5.11}$$

where $(\bar{\mathbf{X}}_1, \bar{\mathbf{\Lambda}}_1) \sim \pi$, $t \in [0,1]$, $\mathrm{d}\bar{t} = -\mathrm{d}t$ *is the negative infinitesimal time step;* $(\bar{\mathbf{B}}_{\bar{t}}^X)_{t\in\mathbb{R}}$ *and* $(\bar{\mathbf{B}}_t^\Lambda)_{t\in\mathbb{R}}$ *are reversed time standard Brownian motions induced by* (5.6)*, and* $\bar{\mathbf{W}}_t^\Lambda \triangleq \mathrm{diag}(\bar{\mathbf{B}}_t^\Lambda)$.

Since $\mathbf{U}$ is no longer involved in (5.11), the boundary condition is only imposed on the joint distribution of $(\mathbf{X}_1, \mathbf{\Lambda}_1)$ such that $\mathrm{law}(\mathbf{X}_1, \mathbf{\Lambda}_1) = \pi$. This assumption implies that the authentic distribution $(\mathbf{X}_0, \mathbf{\Lambda}_0)$ can be recovered from a priori π by the reversed time spectral diffusion SDE. According to score matching techniques [213], we can train two score networks $s_{\boldsymbol{\theta}}(\cdot,\cdot,\cdot)$, $s_{\boldsymbol{\phi}}(\cdot,\cdot,\cdot)$ to learn the score functions $\nabla_{\mathbf{X}}\log p_t(\cdot,\cdot), \nabla_{\mathbf{\Lambda}}\log p_t(\cdot,\cdot)$ via minimizing

$$\begin{aligned} \widehat{\mathcal{E}}(\boldsymbol{\theta}) &\triangleq \mathbb{E}_{\mathbf{G}\sim\mathrm{Unif}(\mathcal{S})}\mathbb{E}_{\mathbf{X}_t|\mathbf{G}}\|s_{\boldsymbol{\theta}}(\mathbf{X}_t,\mathbf{\Lambda}_t,\mathbf{U}_0) - \nabla\log p_{t|0}(\mathbf{X}_t|\mathbf{X}_0)\|^2, \\ \widehat{\mathcal{E}}(\boldsymbol{\phi}) &\triangleq \mathbb{E}_{\mathbf{G}\sim\mathrm{Unif}(\mathcal{S})}\mathbb{E}_{\mathbf{\Lambda}_t|\mathbf{G}}\|s_{\boldsymbol{\phi}}(\mathbf{X}_t,\mathbf{\Lambda}_t,\mathbf{U}_0) - \nabla\log p_{t|0}(\mathbf{\Lambda}_t|\mathbf{\Lambda}_0)\|^2. \end{aligned}$$

In a nutshell, the proposed GSDM has two main steps:

1. Run forward spectral diffusion model on $(\mathbf{X}, \mathbf{\Lambda})$ by (5.10). Train two score networks $s_{\boldsymbol{\theta}}(\cdot,\cdot,\cdot)$, $s_{\boldsymbol{\phi}}(\cdot,\cdot,\cdot)$ to learn score functions showing up in (5.11).

2. Uniformly sample $\widehat{\mathbf{U}}_0$ from the observed eigenvector matrices. Generate plausible $(\widehat{\mathbf{X}}_0, \widehat{\mathbf{\Lambda}}_0)$ by reversing the spectral diffusion SDEs from $t = 1$ to $t = 0$, with estimated score functions $s_{\boldsymbol{\theta}}(\widehat{\mathbf{X}}_t, \widehat{\mathbf{\Lambda}}_t, \widehat{\mathbf{U}}_0), s_{\boldsymbol{\phi}}(\widehat{\mathbf{X}}_t, \widehat{\mathbf{\Lambda}}_t, \widehat{\mathbf{U}}_0)$.

Since the full adjacency matrices are not involved in the computation of diffusion SDEs, GSDM achieves significant acceleration in both training and sampling. Moreover, one can further enhance the computational efficiency, by confining the spectral diffusion to the top-k largest eigenvalues of $\mathbf{A}$, where $k \triangleq [\alpha n]$. Suppose $\mathbf{\Lambda}^k$ is the truncated eigenvalue matrix, where only the top-k diagonal entries of $\mathbf{\Lambda}$ are preserved, we can

define the α-quantile GSDM by substituting the occurrence of $\mathbf{\Lambda}$ in GSDM with $\mathbf{\Lambda}^{(k)}$. As will be seen in the following section, α-quantile GSDM exhibits evidently faster processing speed and comparable performance to GSDM.

The pseudo codes of training and sampling with GSDM are described in Algorithm 5 and Algorithm 6. Specifically, we first train a GSDM from real data by minimizing score-matching errors. On top of that, we are able to generate graph features and eigenvalues of graph adjacency matrices by reversing the forward spectral SDE. By uniformly sampling eigenvectors from the training set, we can construct a plausible graph adjacency matrix via spectral composition.

Algorithm 5: Training of GSDM

Input: Score networks $\mathbf{s}_{\boldsymbol{\theta},t}(\cdot,\cdot,\cdot)$, $\mathbf{s}_{\boldsymbol{\phi},t}(\cdot,\cdot,\cdot)$, maximal diffusion time T, drift functions $\mathbf{f}^X(\cdot,t), \mathbf{f}^\Lambda(\cdot,t)$, noise schedules $\sigma_{X,t}, \sigma_{\Lambda,t}$, learning rate η, training epochs K.

Output: Optimized score network parameters $\boldsymbol{\theta}_K, \boldsymbol{\phi}_K$.

Initialize $\boldsymbol{\theta}_0, \boldsymbol{\phi}_0$
for $k = 1$ **to** K **do**
  $(\mathbf{X}_0, \mathbf{A}_0) \sim \mathcal{G}$;
  $\mathbf{\Lambda}_0, \mathbf{U}_0 \leftarrow \text{EigenDecomposition}(\mathbf{A}_0)$;
  $t \sim \text{Unif}([0,T])$;
  $\mathbf{X}_t \sim \int_0^t f^X(\mathbf{X}_\tau,\tau)\mathrm{d}\tau + \int_0^t \sigma_{X,\tau}\mathrm{d}\mathbf{B}_\tau^X$
  $\mathbf{\Lambda}_t \sim \int_0^t f^\Lambda(\mathbf{\Lambda}_\tau,\tau)\mathrm{d}\tau + \int_0^t \sigma_{\Lambda,\tau}\mathrm{d}\mathbf{B}_\tau^\Lambda$;
  $\widehat{\mathcal{E}}(\boldsymbol{\theta}_k) \leftarrow \|s_{\boldsymbol{\theta}_k}(\mathbf{X}_t, \mathbf{\Lambda}_t, \mathbf{U}_0) - \nabla \log p_{t|0}(\mathbf{X}_t|\mathbf{X}_0)\|^2$;
  $\widehat{\mathcal{E}}(\boldsymbol{\phi}_k) \leftarrow \|s_{\boldsymbol{\phi}_k}(\mathbf{X}_t, \mathbf{\Lambda}_t, \mathbf{U}_0) - \nabla \log p_{t|0}(\mathbf{\Lambda}_t|\mathbf{\Lambda}_0)\|^2$;
  $(\boldsymbol{\theta}_{k+1}, \boldsymbol{\phi}_{k+1}) \leftarrow (\boldsymbol{\theta}_k, \boldsymbol{\phi}_k) - \eta(\nabla\widehat{\mathcal{E}}(\boldsymbol{\theta}_k), \nabla\widehat{\mathcal{E}}(\boldsymbol{\phi}_k))$;
end for
Return: $\boldsymbol{\theta}_K, \boldsymbol{\phi}_K$;

5.3.3 Theoretical Analysis

Here, we provide supportive theoretical evidence for the efficacy of GSDM. In Proposition 5.1, we first study the low-rank structure of the spectral diffusion SDE of adjacency matrices. In Proposition 5.2, we further prove that our proposed spectral diffusion enjoys a sharper reconstruction error bound than the standard graph diffusion model.

Proposition 5.1 (Spectral Diffusion SDEs on Adjacency Matrix)**.** *Suppose* $\mathbf{f}_\Lambda(\mathbf{\Lambda}, t) \triangleq -\sigma_{\Lambda,t}^2/2\mathbf{\Lambda}$. *The spectral diffusion SDE system* (5.10) *induces an* n*-dimensional SDE system of the adjacency matrix* $\mathbf{A}$ *on the full space* $\mathbb{R}^{n\times n}$. *Following previous notations, the forward diffusion SDE is given by*

$$\begin{cases} \mathrm{d}\mathbf{X}_t = & \mathbf{f}^X(\mathbf{X}_t,t)\mathrm{d}t + \sigma_{X,t}\mathrm{d}\mathbf{B}_t^X, \\ \mathrm{d}\mathbf{A}_t = & -\frac{1}{2}\sigma_{\Lambda,t}^2\mathbf{A}_t\mathrm{d}t + \sigma_{\Lambda,t}\mathrm{d}\mathbf{M}_t, \end{cases} \tag{5.12}$$

Algorithm 6: Sampling via GSDM

Input: Score-based models $\mathbf{s}_{\boldsymbol{\theta},t}(\cdot,\cdot,\cdot)$ and $\mathbf{s}_{\boldsymbol{\phi},t}(\cdot,\cdot,\cdot)$, maximal diffusion time T, number of sampling steps M, Langevin-MCMC step sizes $\{\epsilon_i\}_{i=1}^M$, noise schedules $\{\beta_i\}_{i=1}^M$ and priori distribution π. Initialize eigenvectors randomness $\sigma > 0$. Empirical average eigenvectors $\bar{\mathbf{U}}$.

Output: Generated graph data $(\widehat{\mathbf{X}}_0, \widehat{\mathbf{A}}_0)$.

$t \leftarrow T$;

$(\widehat{\mathbf{X}}_T, \widehat{\mathbf{\Lambda}}_T) \sim \pi$

$\widehat{\mathbf{U}}_0 \sim \text{Unif}(\{\mathbf{U} \triangleq \text{EigenVectors}(\mathbf{A}), (\mathbf{X}, \mathbf{A}) \sim \mathcal{G}\})$;

for $m = M-1$ **to** 0 **do**

  $\mathbf{S}_X \leftarrow \mathbf{s}_{\boldsymbol{\theta},t}(\widehat{\mathbf{X}}_t, \widehat{\mathbf{\Lambda}}_t, \widehat{\mathbf{U}}_0)$, $\mathbf{S}_\Lambda \leftarrow \mathbf{s}_{\boldsymbol{\phi},t}(\widehat{\mathbf{X}}_t, \widehat{\mathbf{\Lambda}}_t, \widehat{\mathbf{U}}_0)$

  $t' \leftarrow t - T/(2M)$;

  $\widehat{\mathbf{X}}_{t'} \leftarrow (2 - \sqrt{1-\beta_{m+1}}\widehat{\mathbf{X}}_t + \beta_{m+1}\mathbf{S}_X) + \sqrt{\beta_{m+1}}\mathbf{z}_X$, $\mathbf{z}_X \sim N(\mathbf{0}, \mathbf{I})$ {Prediction step: $\mathbf{X}$}

  $\widehat{\mathbf{\Lambda}}_{t'} \leftarrow (2 - \sqrt{1-\beta_{m+1}}\widehat{\mathbf{\Lambda}}_t + \beta_{m+1}\mathbf{S}_\Lambda) + \sqrt{\beta_{m+1}}\mathbf{z}_\Lambda$, $\mathbf{z}_\Lambda \sim N(\mathbf{0}, \mathbf{I})$; {Prediction step: $\mathbf{\Lambda}$}

  $\mathbf{S}_X \leftarrow \mathbf{s}_{\boldsymbol{\theta},t'}(\widehat{\mathbf{X}}_{t'}, \widehat{\mathbf{\Lambda}}_{t'}, \widehat{\mathbf{U}}_0)$

  $\mathbf{S}_\Lambda \leftarrow \mathbf{s}_{\boldsymbol{\phi},t'}(\widehat{\mathbf{X}}_{t'}, \widehat{\mathbf{\Lambda}}_{t'}, \widehat{\mathbf{U}}_0)$;

  $t \leftarrow t' - T/(2M)$;

  $\widehat{\mathbf{X}}_t \leftarrow \widehat{\mathbf{X}}_{t'} + \epsilon_i \mathbf{S}_X + \sqrt{2\epsilon_i}\boldsymbol{z}_X$, $\mathbf{z}_X \sim N(\mathbf{0}, \mathbf{I})$; {Correction step: $\mathbf{X}$}

  $\widehat{\mathbf{\Lambda}}_t \leftarrow \widehat{\mathbf{\Lambda}}_{t'} + \epsilon_i \mathbf{S}_\Lambda + \sqrt{2\epsilon_i}\mathbf{z}_\Lambda$, $\mathbf{z}_\Lambda \sim N(\mathbf{0}, \mathbf{I})$; {Correction step: $\mathbf{\Lambda}$}

end for

$\widehat{\mathbf{A}}_0 = \widehat{\mathbf{U}}_0 \widehat{\mathbf{\Lambda}}_0 \widehat{\mathbf{U}}_0^{\mathbf{T}}$;

Return: $(\widehat{\mathbf{X}}_0, \widehat{\mathbf{A}}_0)$;

where $(\mathbf{M}_t)_{t\in[0,1]}$ *is a* n*-dimensional centered Gaussian process on* $\mathbb{R}^{n\times n}$*, with zero mean and covariance kernel* $\mathcal{K}(s,t) : [0,1]\times[0,1] \mapsto \mathbb{R}^{n\times n\times n\times n}$ *as*

$$\mathcal{K}(s,t)_{i,j,k,l} = \min(s,t) \cdot \sum_{h=1}^{n} \mathbf{U}_0[i,h]\mathbf{U}_0[j,h]\mathbf{U}_0[k,h]\mathbf{U}_0[l,h].$$

Hence, the conditional distribution of $\mathbf{A}_t$ *on* $\mathbf{A}_0$ *is Gaussian*

$$\mathbf{A}_t|\mathbf{A}_0 \sim N(\mathbf{A}_t; \mathbf{A}_0 e^{-\frac{1}{2}\int_0^t \sigma_\tau^2 \mathrm{d}\tau}, (1 - e^{-\int_0^t \sigma_\tau^2 \mathrm{d}\tau})\mathcal{K}(1,1)), \tag{5.13}$$

which admits a closed-form probability density function.

Remark 5.1. *The proof can be found in Section 5.4.1. Proposition 5.1 shows that our spectral diffusion framework substantially recasts the evolution of the adjacency matrix, by driving the* n^2*-dimensional SDE with* n*-dimensional noise* $(\mathbf{M}_t)_{t\in\mathbb{R}}$*. Guided by the graph spectrum, the diffusion is concentrated to the salient parts of* $\text{supp}(\mathbf{A})$ *to prevent introducing irreducible noise to the out-of-support regions.*

The central question for graph generation is how to measure the quality of the synthesis data that is recovered from noise with the learned score function. The key to this question is to establish the reconstruction error bound, i.e. the expected error between the data reconstructed with ground truth score $\nabla \log p_t(\cdot)$ and the learned scores $s_{\boldsymbol{\phi}}(\cdot)$. For graph topology generation, our GSDM is proven to enjoy a sharper reconstruction bound than standard graph diffusion. Detailed proof can be found in Section 5.4.2.

Proposition 5.2 (Reconstruction Bounds for Adjacency Matrix Generation)**.** *Following the previous notations, we define the estimated reversed time SDEs for standard and spectral graph diffusion models as*

$$\mathrm{d}\widehat{\mathbf{A}}_t^{\text{full}} = \left(-\frac{1}{2}\sigma_t^2 \widehat{\mathbf{A}}_t^{\text{full}} - \sigma_t^2 s_{\boldsymbol{\phi}}(\widehat{\mathbf{A}}_t^{\text{full}}, t)\right)\mathrm{d}\bar{t} + \sigma_t \mathrm{d}\bar{\mathbf{B}}_t^A,$$
$$\mathrm{d}\widehat{\mathbf{A}}_t^{\text{spec}} = \left(-\frac{1}{2}\sigma_t^2 \widehat{\mathbf{A}}_t^{\text{spec}} - \sigma_t^2 s_{\boldsymbol{\varphi}}(\widehat{\mathbf{A}}_t^{\text{spec}}, t)\right)\mathrm{d}\bar{t} + \sigma_t \mathrm{d}\bar{\mathbf{M}}_t,$$

where both generation methods share the same drift term and noise schedule $(\sigma_t)_{t\in[0,1]}$. We further assume that $\|s_{\boldsymbol{\phi}}(\cdot,t)\|_{\text{lip}} = \mathcal{O}(\mathbb{E}_{|\mathbf{A}_0}\|\nabla_{\mathbf{A}} \log p_{t|0}(\widehat{\mathbf{A}}_t^{\text{full}})\|_{\text{lip}})$ and $\|s_{\boldsymbol{\varphi}}(\cdot,t)\|_{\text{lip}} = \mathcal{O}(\mathbb{E}_{|\mathbf{A}_0}\|\nabla_{\mathbf{A}} \log p_{t|0}(\widehat{\mathbf{A}}_t^{\text{spec}})\|_{\text{lip}})$ holds almost surly. By reversing both SDEs from $t = 1$ to $t = 0$, we obtain $\widehat{\mathbf{A}}_t^{\text{full}}$ and $\widehat{\mathbf{A}}_t^{\text{spec}}$, which are two reconstructions of the authentic $\mathbf{A}_0$, with expected reconstruction errors bounded by

$$\begin{aligned}
&\mathbb{E}\|\mathbf{A}_0 - \widehat{\mathbf{A}}_0^{\text{full}}\|^2 \\
&\leqslant M\mathcal{E}(\boldsymbol{\phi}) \cdot \left(1 + n^2 K \int_0^1 \Sigma_t^{-2} \exp\left(n^2 K \int_t^1 \Sigma_s^{-2}\mathrm{d}s\right)\mathrm{d}t\right), \\
&\mathbb{E}\|\mathbf{A}_0 - \widehat{\mathbf{A}}_0^{\text{spec}}\|^2 \\
&\leqslant M\mathcal{E}(\boldsymbol{\varphi}) \cdot \left(1 + nK \int_0^1 \Sigma_t^{-2} \exp\left(nK \int_t^1 \Sigma_s^{-2}\mathrm{d}s\right)\mathrm{d}t\right),
\end{aligned}$$

where $M \triangleq C^2\|\sigma_\cdot\|_\infty^4$; $K \triangleq 2ML/\mathbb{E}\|\mathbf{A}_0\|_{2,2}$; C, L are absolute constants; $\Sigma_t^2 \triangleq 1 - e^{-\int_0^t \sigma_s^2 \mathrm{d}s}$; $\mathcal{E}(\cdot)$ is the expected score matching objective defined in (5.3).

Remark 5.2. *While the error bound of $\widehat{\mathbf{A}}_0^{\text{full}}$ is at the order of $\mathcal{O}(n^2\exp(n^2))$, $\widehat{\mathbf{A}}_0^{\text{spec}}$ exhibits a much sharper bound of order $\mathcal{O}(n\exp(n))$. For large-scale graphs with a large number of nodes n, our proposed GSDM enjoys a substantially better performance guarantee, which coincides with our numerical results. Moreover, under additional conditions, the error bounds can be significantly improved for well-trained score networks $\boldsymbol{\phi}^*, \boldsymbol{\varphi}^*$, by showing that $\mathcal{E}(\boldsymbol{\phi}^*) \ll \mathcal{E}(\boldsymbol{\phi})$ as in Proposition 5.3. The detailed proof can be found in Section 5.4.3.*

Definition 5.4 (β-smooth)**.** *$f : \mathbb{R}^m \mapsto \mathbb{R}^n$ is called β-smooth if and only if*

$$\|f(x_1) - f(x_2) - \nabla f(x_2)^\top (x_1 - x_2)\| \leqslant \frac{\beta}{2}\|x_1 - x_2\|^2$$

holds for $\forall\ x_1, x_2 \in \mathbb{R}^m$.

Proposition 5.3 (Convergence of Score Matching Objective Minimization)**.** *Recall that the score matching objective of a generic sample* $\mathbf{Z} \in \mathbb{R}^d$ *(*$\widehat{\mathbf{A}}^{\text{full}}$ *or* $\widehat{\mathbf{A}}^{\text{spec}}$*) is defined as*

$$\mathcal{E}(\boldsymbol{\theta}; \mathbf{Z}) \triangleq \mathbb{E}_{\mathbf{Z}_t|\mathbf{Z}} \|s_{\boldsymbol{\theta}}(\mathbf{Z}_t, t) - \nabla_{\mathbf{Z}} \log p_t(\mathbf{Z}_t)\|^2, \tag{5.14}$$

and expected and empirical score-matching errors are defined as

$$\mathcal{E}(\boldsymbol{\theta}) \triangleq \mathbb{E}_{\mathbf{Z}\sim\mathcal{D}} \mathcal{E}(\boldsymbol{\theta}; \mathbf{Z}) \tag{5.15}$$

$$\widehat{\mathcal{E}}(\boldsymbol{\theta}; \mathcal{S}_N) \triangleq \mathbb{E}_{\mathbf{Z}\sim\mathcal{S}_N} \mathcal{E}(\boldsymbol{\theta}; \mathbf{Z}) \tag{5.16}$$

where $\mathcal{D}$ *is the population distribution of* $\mathbf{Z}$*, and* $\mathcal{S} \triangleq \{\mathbf{Z}^i\}_{i=1}^N$ *is the uniform distribution over an i.i.d sampled training dataset of size* N*. Suppose* $\mathcal{E}(\boldsymbol{\theta})$ *is minimized via running standard Stochastic Gradient Descent (SGD) on training data, i.e. at the* k*-th iteration,* $\boldsymbol{\theta}_k$ *is updated on a mini-batch of size* b

$$\boldsymbol{\theta}_{k+1} \triangleq \boldsymbol{\theta}_k - \eta \nabla_{\boldsymbol{\theta}} \widehat{\mathcal{E}}(\boldsymbol{\theta}; \mathcal{S}_b). \tag{5.17}$$

Assume that almost surely (w.r.t $\mathbf{Z}$*),* $\mathcal{E}(\cdot; \mathbf{Z})$ *is* β*-smooth, and the tangent kernel* $\mathbf{K}_{\boldsymbol{\theta}}(\mathcal{S}) \in \mathbb{R}^{Nd\times Nd}$ *of* $s_{\boldsymbol{\theta}}(\cdot)$ *satisfies*

$$\begin{aligned}
&\lambda_{\min}\left(\mathbf{K}_{\boldsymbol{\theta}}(\mathcal{S})\right) \geqslant \lambda > 0, \ \boldsymbol{\theta} \in B(\boldsymbol{\theta}_0, R), \\
&\mathbf{K}_{\boldsymbol{\theta}}(\mathcal{S})[Ni+1 : N(i+1), Nj+1 : N(j+1)] \\
&\triangleq \nabla_{\boldsymbol{\theta}} s_{\boldsymbol{\theta}}(\mathbf{Z}^i)^\top \nabla_{\boldsymbol{\theta}} s_{\boldsymbol{\theta}}(\mathbf{Z}^j),
\end{aligned}$$

with $R = 2N\sqrt{2\beta\mathcal{E}(\boldsymbol{\theta}_0)}/(\mu\delta)$*,* $\delta > 0$*. Then, with probability* $1-\delta$ *over the choice of mini-batch* $\mathcal{S}_b$*, SGD with a learning rate* $\eta \leqslant \frac{\lambda/N}{N\beta(N^2\beta+\lambda(b-1)/N)}$ *converges to a global solution in the ball* $B(\boldsymbol{\theta}_0, R)$ *with exponential convergence rate*

$$\mathcal{E}(\boldsymbol{\theta}_k) \leqslant \left(1 - \frac{\lambda b \eta}{N}\right)^k \mathcal{E}(\boldsymbol{\theta}_0). \tag{5.18}$$

5.4 Detailed Proofs

5.4.1 Proof of Proposition 5.1

Proof: Plugging $\mathbf{A}_t = \mathbf{U}_0 \boldsymbol{\Lambda}_t \mathbf{U}_0^\top$ into (5.12) yields

$$\begin{aligned}
\mathrm{d}\mathbf{A}_t =& -\frac{1}{2}\sigma_{\Lambda,t}^2 \mathbf{A}_t \mathrm{d}t + \sigma_{\Lambda,t} \mathrm{d}\mathbf{M}_t, \\
\mathbf{M}_t \triangleq& \mathbf{U}_0 \mathbf{W}_t^\Lambda \mathbf{U}_0^\top = \mathbf{U}_0 \cdot \mathrm{diag}(\mathbf{B}_t^\Lambda) \cdot \mathbf{U}_0^\top.
\end{aligned}$$

Since $(\mathbf{M}_t)_{t\in\mathbb{R}}$ is a linear transformation of $(\mathbf{B}_t^\Lambda)_{t\in\mathbb{R}}$, the standard Brownian motion on $\mathbb{R}^n$, hence $(\mathbf{M}_t)_{t\in\mathbb{R}}$ is a rank-n centered Gaussian process in $\mathbb{R}^{n\times n}$. Moreover, $\mathbf{M}_t$ is

characterized by its covariance kernel $\mathcal{K}(s,t):[0,1]\times[0,1]\mapsto\mathbb{R}^{n\times n\times n\times n}$, which is given by

$$\begin{aligned}
&\mathcal{K}(s,t)_{i,j,k,l}\\
&\triangleq \mathrm{Cov}(\mathbf{M}_s[i,j],\mathbf{M}_t[k,l])=\mathbb{E}[\mathbf{M}_s[i,j]\mathbf{M}_t[k,l]]\\
&=\mathbb{E}\left(\sum_{h=1}^{n}\mathbf{U}_0[i,h]\mathbf{B}_s^{\Lambda}[h]\mathbf{U}_0[j,h]\right)\left(\sum_{h=1}^{n}\mathbf{U}_0[k,h]\mathbf{B}_t^{\Lambda}[h]\mathbf{U}_0[l,h]\right)\\
&=\sum_{h=1}^{n}\mathbb{E}(\mathbf{B}_s^{\Lambda}[h]\mathbf{B}_t^{\Lambda}[h])\mathbf{U}_0[i,h]\mathbf{U}_0[j,h]\mathbf{U}_0[k,h]\mathbf{U}_0[l,h]\\
&=\min(s,t)\sum_{h=1}^{n}\mathbf{U}_0[i,h]\mathbf{U}_0[j,h]\mathbf{U}_0[k,h]\mathbf{U}_0[l,h].
\end{aligned}$$

Notice that (5.10) is an Ornstein–Uhlenbeck process, which admits a closed-form solution

$$\mathbf{\Lambda}_t=\mathbf{\Lambda}_0e^{-\frac{1}{2}\int_0^t\sigma_\tau^2\mathrm{d}\tau}+(1-e^{-\int_0^t\sigma_\tau^2\mathrm{d}\tau})\mathbf{W}_1^{\Lambda}. \tag{5.19}$$

Hence, plugging (5.19) into $\mathbf{A}_t=\mathbf{U}_0\mathbf{\Lambda}_t\mathbf{U}_0^\top$ yields

$$\mathbf{A}_t=\mathbf{A}_0e^{-\frac{1}{2}\int_0^t\sigma_\tau^2\mathrm{d}\tau}+(1-e^{-\int_0^t\sigma_\tau^2\mathrm{d}\tau})\mathbf{M}_1. \tag{5.20}$$

This completes the proof.

5.4.2 Proof of Proposition 5.2

In preparation for the main proof, we first establish some technical lemmas.

Lemma 5.2 (Reconstruction Bound for Generic Diffusion)**.** *We first consider the following oracle reversed time SDE on* $(\mathbb{R}^d,\|\cdot\|)$

$$\mathrm{d}\bar{\mathbf{Z}}_t=\left(\mathbf{f}(\bar{\mathbf{Z}}_t,t)-\sigma_t^2\nabla_{\mathbf{Z}}\log p_t(\bar{\mathbf{Z}}_t)\right)\mathrm{d}\bar{t}+\sigma_t\mathrm{d}\bar{\mathbf{B}}_t,\ t\in[0,1],$$

and we define the corresponding estimated reverse time SDE as

$$\mathrm{d}\widehat{\mathbf{Z}}_t=\left(\mathbf{f}(\widehat{\mathbf{Z}}_t,t)-\sigma_t^2s_{\boldsymbol{\phi}}(\widehat{\mathbf{Z}}_t,t)\right)\mathrm{d}\bar{t}+\sigma_t\mathrm{d}\bar{\mathbf{B}}_t,\ t\in[0,t],$$

where $s_{\boldsymbol{\phi}}(\cdot)$ *is optimized to predict the Stein score function* $\nabla_{\mathbf{Z}}\log p_t(\mathbf{Z}_t)$ *by minimizing the score matching objective*

$$\min_{\boldsymbol{\phi}}\mathcal{E}(\boldsymbol{\phi})\triangleq\mathbb{E}_{\mathbf{Z}}\mathbb{E}_{\mathbf{Z}_t|\mathbf{Z}}\|s_{\boldsymbol{\phi}}(\mathbf{Z}_t,t)-\nabla_{\mathbf{Z}}\log p_t(\mathbf{Z}_t)\|^2.$$

Then for any $\boldsymbol{\phi}$*, the construction error is bounded by*

$$\begin{aligned}
\mathbb{E}\|\mathbf{Z}_0-\widehat{\mathbf{Z}}_0\|^2\leqslant&C^2\|\sigma_\cdot\|_\infty^4\mathcal{E}(\boldsymbol{\phi})\\
&\cdot\left(1+\int_0^1F(t)\exp\left(\int_t^1F(s)\mathrm{d}s\right)\mathrm{d}t\right),
\end{aligned} \tag{5.21}$$

where $F(t)\triangleq C^2\sigma_t^4\|s_{\boldsymbol{\phi}}(\cdot,t)\|_{\mathrm{lip}}^2+C\|\mathbf{f}(\cdot,t)\|_{\mathrm{lip}}^2$ *and* C *is a constant.*

Proof: [Proof of Proposition 5.2] Assumptions of Proposition 5.2 imply

$$\begin{aligned}\|\mathbf{f}(\cdot,t)\|_{\text{lip}} =& \frac{1}{2}\sigma_t^2,\\ \|s_{\boldsymbol{\phi}}(\cdot,t)\|_{\text{lip}} \leqslant& L\cdot\mathbb{E}_{|\mathbf{A}_0}\|\nabla\log p_{t|0}(\widehat{\mathbf{A}}_t^{\text{full}})\|_{\text{lip}},\\ \|s_{\boldsymbol{\varphi}}(\cdot,t)\|_{\text{lip}} \leqslant& L\cdot\mathbb{E}_{|\mathbf{A}_0}\|\nabla\log p_{t|0}(\widehat{\mathbf{A}}_t^{\text{spec}})\|_{\text{lip}},\end{aligned}$$

where L is a constant. According to [187, 188], at each time step $t\in[0,1]$, the oracle conditional score function $\nabla\log p_{t|0}(\cdot)$ is equivalent to a mapping that maps the input d-dimensional random variable $\mathbf{Z}_t$ to $\Sigma_t^{-1}\boldsymbol{\varepsilon}$. The shown up $\boldsymbol{\varepsilon}$ is a d-dimensional standard Gaussian vector that is independent to $\mathbf{Z}_t$, and Σ_t is the standard deviation of $\mathbf{Z}_t$, which is given by $\Sigma_t \triangleq \left(1-e^{-\int_0^t\sigma_s^2\mathrm{d}s}\right)^{\frac{1}{2}}$. Thus, the expected Lipschitz norm of oracle conditional score function is bounded by

$$\begin{aligned}\left(\mathbb{E}_{|\mathbf{A}_0}\|\nabla\log p_{t|0}(\mathbf{Z}_t)\|_{\text{lip}}\right)^2 &\leqslant \mathbb{E}_{|\mathbf{A}_0}\|\nabla\log p_{t|0}(\mathbf{Z}_t)\|_{\text{lip}}^2\\ &\leqslant \frac{\Sigma_t^{-2}}{\|\mathbf{Z}_0\|^2}\mathbb{E}\|\boldsymbol{\varepsilon}\|^2 = \frac{\Sigma_t^{-2}d}{\|\mathbf{Z}_0\|^2}.\end{aligned}\tag{5.22}$$

With Lemma 5.2 in hand, we only need to plug (5.22) into (5.21), by substituting the notations $(\mathbf{Z},d,\|\cdot\|)$ with $(\widehat{\mathbf{A}}_t^{\text{full}},n^2,\|\cdot\|_{2,2})$ and $(\widehat{\boldsymbol{\Lambda}}_t^{\text{spec}},n,\|\cdot\|_{2,2})$ to bound the Lipschitz norm of the score networks. This leads us to

$$\begin{aligned}&\mathbb{E}\|\mathbf{A}_0-\widehat{\mathbf{A}}_0^{\text{full}}\|^2\\ &\leqslant M\mathcal{E}(\boldsymbol{\phi})\cdot\left(1+n^2K\cdot\int_0^1\Sigma_t^{-2}\exp\left(n^2K\int_t^1\Sigma_s^{-2}\mathrm{d}s\right)\mathrm{d}t\right),\\ &\mathbb{E}\|\boldsymbol{\Lambda}_0-\widehat{\boldsymbol{\Lambda}}_0^{\text{spec}}\|^2\\ &\leqslant M\mathcal{E}(\boldsymbol{\phi})\cdot\left(1+nK\cdot\int_0^1\Sigma_t^{-2}\exp\left(nK\int_t^1\Sigma_s^{-2}\mathrm{d}s\right)\mathrm{d}t\right),\end{aligned}$$

where $M\triangleq C^2\|\sigma_\cdot\|_\infty^4$ and $K\triangleq 2ML/\mathbb{E}\|\mathbf{A}_0\|_{2,2}$. For the spectral diffusion part, the final proof step is completed by the fact that

$$\begin{aligned}&\mathbb{E}\|\mathbf{A}_0-\widehat{\mathbf{A}}_0^{\text{spec}}\|^2=\mathbb{E}\|\mathbf{U}_0\left(\boldsymbol{\Lambda}_0-\widehat{\boldsymbol{\Lambda}}_0^{\text{spec}}\right)\mathbf{U}_0^\top\|^2=\mathbb{E}\|\boldsymbol{\Lambda}_0-\widehat{\boldsymbol{\Lambda}}_0^{\text{spec}}\|^2,\\ &\mathbb{E}\|\mathbf{A}_0\|_{2,2}=\mathbb{E}\|\mathbf{U}_0\boldsymbol{\Lambda}_0\mathbf{U}_0^\top\|_{2,2}=\mathbb{E}\|\boldsymbol{\Lambda}_0\|_{2,2}.\end{aligned}$$

This completes the proof.

Proof: [Proof of Lemma 5.2]

To bound the expected reconstruction error $\mathbb{E}\|\mathbf{Z}_0-\widehat{\mathbf{Z}}_0\|^2$, we first analysis how $\mathbb{E}\|\bar{\mathbf{Z}}_t-\widehat{\mathbf{Z}}_t\|^2$ evolves as time t is reversed from 1 to 0. By definition, it holds that

$$\bar{\mathbf{Z}}_t-\widehat{\mathbf{Z}}_t=\int_1^t\Big(\mathbf{f}(\bar{\mathbf{Z}}_s,s)-\mathbf{f}(\widehat{\mathbf{Z}}_s,s)\Big)+\sigma_s^2\Big(s_{\boldsymbol{\phi}}(\widehat{\mathbf{Z}}_s,s)-\nabla_{\mathbf{Z}}\log p_s(\bar{\mathbf{Z}}_s)\Big)\,\mathrm{d}\bar{s}.$$

By taking norm and expectation on both sides, we have

$$
\begin{aligned}
&\mathbb{E}\|\bar{\mathbf{Z}}_t - \widehat{\mathbf{Z}}_t\|^2 \\
&\leqslant \mathbb{E}\int_1^t \left\| \Big(\mathbf{f}(\bar{\mathbf{Z}}_s, s) - \mathbf{f}(\widehat{\mathbf{Z}}_s, s)\Big) + \sigma_s^2 \Big(s_{\boldsymbol{\phi}}(\widehat{\mathbf{Z}}_s, s) - \nabla_{\mathbf{Z}} \log p_s(\bar{\mathbf{Z}}_s) \Big) \right\|^2 \mathrm{d}\bar{s} \\
&\leqslant C\mathbb{E}\int_1^t \left\| \mathbf{f}(\bar{\mathbf{Z}}_s, s) - \mathbf{f}(\widehat{\mathbf{Z}}_s, s) \right\|^2 \mathrm{d}\bar{s} \\
&\quad + C\mathbb{E}\int_1^t \sigma_s^4 \left\| s_{\boldsymbol{\phi}}(\widehat{\mathbf{Z}}_s, s) - \nabla_{\mathbf{Z}} \log p_s(\bar{\mathbf{Z}}_s) \right\|^2 \mathrm{d}\bar{s} \\
&\leqslant C^2 \int_1^t \sigma_s^4 \cdot \mathbb{E}\left\| s_{\boldsymbol{\phi}}(\widehat{\mathbf{Z}}_s, s) - s_{\boldsymbol{\phi}}(\bar{\mathbf{Z}}_s, s) \right\|^2 \\
&\quad + \sigma_s^4 \cdot \mathbb{E}\left\| s_{\boldsymbol{\phi}}(\bar{\mathbf{Z}}_s, s) - \nabla_{\mathbf{Z}} \log p_s(\bar{\mathbf{Z}}_s) \right\|^2 \mathrm{d}\bar{s} \\
&\quad + C\int_1^t \|\mathbf{f}(\cdot, s)\|_{\mathrm{lip}}^2 \cdot \mathbb{E}\left\| \bar{\mathbf{Z}}_s - \widehat{\mathbf{Z}}_s \right\|^2 \mathrm{d}\bar{s} \\
&\leqslant \int_1^t \underbrace{\Big(C^2\sigma_s^4 \|s_{\boldsymbol{\phi}}(\cdot, s)\|_{\mathrm{lip}}^2 + C\|\mathbf{f}(\cdot, s)\|_{\mathrm{lip}}^2 \Big)}_{F(s)} \cdot \mathbb{E}\left\| \widehat{\mathbf{Z}}_s - \bar{\mathbf{Z}}_s \right\|^2 \mathrm{d}\bar{s} \\
&\quad + \underbrace{C^2 \int_1^t \sigma_s^4 \mathrm{d}\bar{s} \cdot \mathcal{E}(\boldsymbol{\phi})}_{G(t)}.
\end{aligned}
$$

The proof is completed by applying the Grönwall's inequality to $t \mapsto \mathbb{E}\|\bar{\mathbf{Z}}_t - \widehat{\mathbf{Z}}_t\|^2$, which yields

$$
\begin{aligned}
&\mathbb{E}\|\mathbf{Z}_0 - \widehat{\mathbf{Z}}_0\|^2 \\
&\leqslant G(0) + \int_0^1 G(t)F(t) \exp\left(\int_t^1 F(s)\mathrm{d}s \right) \mathrm{d}t \\
&\leqslant C^2 \|\sigma_\cdot\|_\infty^4 \mathcal{E}(\boldsymbol{\phi}) \cdot \left(1 + \int_0^1 F(t) \exp\left(\int_t^1 F(s)\mathrm{d}s \right) \mathrm{d}t \right).
\end{aligned}
$$

5.4.3 Proof of Proposition 5.3

Proof: Since $\mathcal{E}(\boldsymbol{\theta}) = \mathbb{E}_{\mathcal{S}\sim\mathcal{D}^N} \widehat{\mathcal{E}}(\boldsymbol{\theta}; \mathcal{S}_N)$, we only need to bound the empirical risk $\widehat{\mathcal{E}}(\boldsymbol{\theta}; \mathcal{S}_N)$. By assumption, denote

$$
\mathbf{h} \triangleq \begin{pmatrix} s_{\boldsymbol{\theta}}(\mathbf{Z}_0^1) - \nabla \log p(\mathbf{Z}_0^1 | \mathbf{Z}_1^1)^\top \\ \vdots \\ s_{\boldsymbol{\theta}}(\mathbf{Z}_0^N) - \nabla \log p(\mathbf{Z}_0^N | \mathbf{Z}_1^N)^\top \end{pmatrix} \in \mathbb{R}^{Nd\times 1},
$$

It holds that

$$
\|\nabla_{\boldsymbol{\theta}} \widehat{\mathcal{E}}(\boldsymbol{\theta}; \mathcal{S}_N)\|^2 = \frac{1}{N^2} \mathbf{h}^\top \mathbf{K}_{\boldsymbol{\theta}}(\mathcal{S}) \mathbf{h} \geqslant \frac{\lambda}{N^2} \|\mathbf{h}\|^2 = \frac{\lambda}{N} \widehat{\mathcal{E}}(\boldsymbol{\theta}; \mathcal{S}_N),
$$

which implies that $\widehat{\mathcal{E}}(\boldsymbol{\theta}; \mathcal{S}_N)$ satisfies the $\frac{\lambda}{N}$-Polyak Łojasiewicz condition [124]. Then the proof is completed by applying Theorem 7 in [124] to $\widehat{\mathcal{E}}(\cdot; \mathcal{S}_N)$.

5.5 Experiments

In this section, we evaluate our proposed GSDM with state-of-the-art graph generative baselines on two types of graph generation tasks: generic graph generation and molecule generation, over several benchmark datasets. We also conduct extensive ablation studies and visualizations to further illustrate the effectiveness and efficiency of GSDM.

5.5.1 Generic Graph Generation

Datasets We test our model on three generic datasets with different scales, and we use N to denote the number of nodes: (1) Community-small ($12 \leq N \leq 20$): contains 100 small community graphs. (2) Enzymes ($10 \leq N \leq 125$): contains 578 protein graphs which represent the protein tertiary structures of the enzymes from the BRENDA database. (3) Grid ($100 \leq N \leq 400$): contains 100 standard 2D grid graphs. To fairly compare our model with baselines, we adopt the experimental and evaluation setting from [238] with the same train/test split.

Baselines We compare our model with well-known or state-of-the-art graph generation methods, which can be categorized into auto-regressive models and one-shot models. The **auto-regressive model** refers to sequential generation, which constructs a graph via a set of consecutive steps, usually done nodes by nodes and edges by edges [64]. Under this category, we include DeepGMG [119], GraphRNN [238], GraphAF [183], and GraphDF [139]. The **one-shot model** refers to building a probabilistic graph model that can generate all nodes and edges in one shot [64]. Under this category, we include GraphVAE [186], Graph Normalizing Flow(GNF) [127], EDP-GNN [153], and GDSS [91]. In addition to graph generation methods, we incorporate two SOTA diffusion-based generation methods from the field of computer vision into the task of graph generation: SubspaceDiff [89] and WSGM [65].

Metrics We adopt the maximum mean discrepancy (MMD) to compare the distributions of graph statistics, such as the degree, the clustering coefficient, and the number of occurrences of orbits with 4 nodes [91, 238] between the same number of generated and test graphs.

Results We show the results in Table 5.1. The results show that our proposed GSDM significantly outperforms both the auto-regressive baselines and one-shot baseline methods. Specifically, GDSS is the SOTA graph diffusion model which performs the diffusion in the whole space of the graph data. Our method substantially outperforms GDSS in terms of both average performance and convergence rate (Figure 5.3), which demonstrates the advantages of performing SDEs in the spectrum of the graph compared to the whole space.

Table 5.1: **Generation results on the generic graph datasets.** We report the MMD distances between the test datasets and generated graphs. The best results are highlighted in bold (the smaller the better). Hyphen (-) denotes out-of-resources that take more than 10 days or are not applicable due to memory issues.

	Community-small				Enzymes				Grid			
	Synthetic, $12 \leq \|V\| \leq 20$				Real, $10 \leq \|V\| \leq 125$				Synthetic, $100 \leq \|V\| \leq 400$			
	Deg.↓	Clus.↓	Orbit↓	Avg.↓	Deg.↓	Clus.↓	Orbit↓	Avg.↓	Deg.↓	Clus.↓	Orbit↓	Avg.↓
DeepGMG [119]	0.220	0.950	0.400	0.523	-	-	-	-	-	-	-	-
GraphRNN [238]	0.080	0.120	0.040	0.080	0.017	**0.043**	0.021	0.043				
GraphAF [183]	0.18	0.200	0.020	0.133	1.669	1.283	0.266	1.073	-	-	-	-
GraphDF [139]	0.060	0.120	0.030	0.070	1.503	1.061	0.202	0.922	-	-	-	-
GraphVAE [186]	0.350	0.980	0.540	0.623	1.369	0.629	0.191	0.730	1.619	**0.0**	0.919	0.846
GNF [127]	0.200	0.200	0.110	0.170	-	-	-	-	-	-	-	-
EDP-GNN [153]	0.053	0.144	0.026	0.074	0.023	0.268	0.082	0.124	0.455	0.238	0.328	0.340
SubspaceDiff [89]	0.057	0.098	0.012	0.056	0.037	0.099	0.018	0.051	0.124	0.013	0.090	0.076
WSGM [65]	0.039	0.084	0.009	0.044	0.034	0.097	0.013	0.048	0.083	0.006	0.065	0.051
GDSS [91]	0.045	0.086	0.007	0.046	0.026	0.102	0.009	0.046	0.111	0.005	0.070	0.062
Ours	**0.011**	**0.015**	**0.001**	**0.009**	**0.013**	0.088	**0.009**	**0.037**	**0.002**	**0.0**	**0.0**	**0.0007**

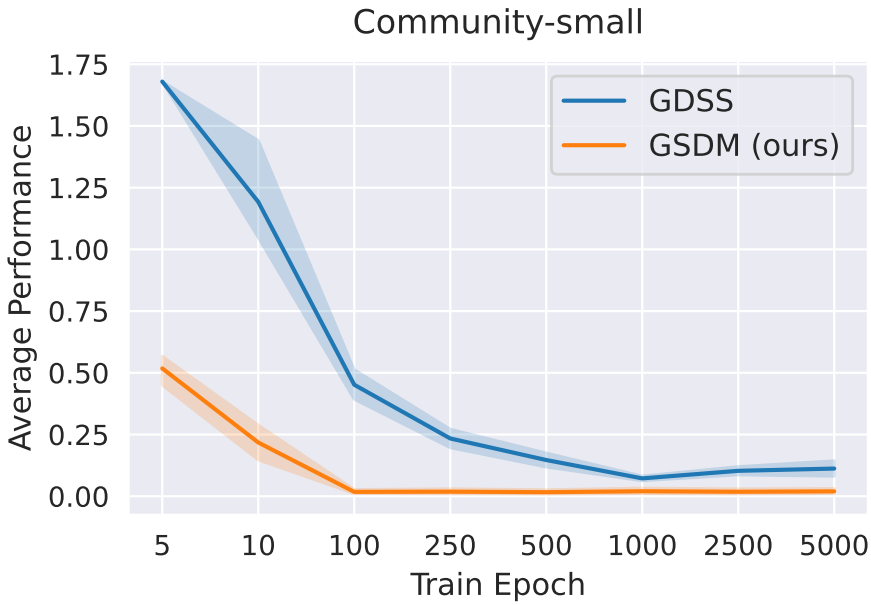


Fig. 5.3: GSDM enjoys a significantly faster convergence rate than GDSS. On the community-small dataset, our GSDM reaches the SOTA performance within 100 training epochs.

5.5.2 Molecules Generation

Besides generic graph generation, our model can also generate organic molecules through our proposed reverse diffusion process. We test our model with two well-known molecule datasets: QM9 [167] and ZINC250K [86]. Following previous works [91,139], the molecules

Table 5.2: **Generation results on the QM9 and ZINC250k datasets.** Results are the means of three different runs, and the best results are highlighted in bold. Values denoted by * are taken from the respective original papers. Other results are obtained by running open-source codes. Val. w/o corr. denotes the Validity w/o correction metric, and values that do not exceed 50% are underlined.

	QM9					ZINC250k				
Method	Validity (%)↑	Val. w/o corr. (%)↑	NSPDK↓	FCD↓	Time (s)↓	Validity (%)↑	Val. w/o corr. (%)↑	NSPDK↓	FCD↓	Time (s)↓
GraphAF [183]	100	67*	0.020	5.268	$2.28e^3$	100	68*	0.044	16.289	$5.72e^3$
GraphAF+FC	100	74.43	0.021	5.625	$2.32e^3$	100	68.47	0.044	16.023	$5.91e^3$
GraphDF [139]	100	82.67*	0.063	10.816	$5.08e^4$	100	89.03*	0.176	34.202	$5.87e^4$
GraphDF+FC	100	93.88	0.064	10.928	$4.72e^4$	100	90.61	0.177	33.546	$5.79e^4$
MoFlow [245]	100	91.36	0.017	4.467	**4.58**	100	63.11	0.046	20.931	**25.9**
EDP-GNN [153]	100	47.52	0.005	2.680	$4.13e^3$	100	82.97	0.049	16.737	$8.41e^3$
GraphEBM [131]	100	8.22	0.030	6.143	35.33	100	5.29	0.212	35.471	53.72
GDSS [91]	100	95.72*	**0.003***	2.900*	$1.06e^2$	100	**97.01***	0.019*	14.656*	$2.11e^3$
Ours	100	**99.9**	**0.003**	**2.650**	18.02	100	92.70	**0.017**	**12.956**	45.91

are kekulized by the RDKit library [111] with hydrogen atoms removed. We evaluate the quality of 10,000 generated molecules with **Frechet ChemNet Distance (FCD)** [164], **Neighborhood subgraph pairwise distance kernel (NSPDK) MMD** [27], **validity w/o correction**, and the **generation time**. FCD computes the distance between the testing and the generated molecules using the activations of the penultimate layer of the ChemNet. (NSPDK) MMD computes the MMD between the generated and the testing set which takes into account both the node and edge features for evaluation. Generally speaking, FCD measures the generation quality in the view of molecules in the chemical space, while NSPDK MMD evaluates the generation quality from the graph structure perspective. Besides, following [91], we also include the **validity w/o correction** as another metric to explicitly evaluate the quality of molecule generation prior to the correction procedure. It computes the fraction of the number of valid molecules without valency correction or edge resampling over the total number of generated molecules. In contrast, **validity** measures the fraction of the valid molecules after the correction phase. **Generation time** measures the time for generating 10,000 molecules in the form of RDKit. It is a salient measure of the practicability of molecule generation, especially for macromolecule generation.

Baselines We compare our model with the state-of-the-art molecule generation models. The baselines include SOTA auto-regressive models: GraphAF [183] is a flow-based model, and GraphDF [139] is a flow-based model using discrete latent variables. Following GDSS [91], we modify the architecture of GraphAF and GraphDF to consider formal charges in the molecule generation, denoted as GraphAF+FC and GraphDF+FC, for fair comparisons. For the one-shot model, we include MoFlow [245], which is a flow-based model; EDP-GNN [153] and GDSS [153] which are both diffusion models.

Results We show the results in Table 5.2. Evidently, GSDM achieves the highest performance under most of the metrics. The highest scores in NSPDK and FCD show that GSDM is able to generate molecules that have close data distributions to the real molecules in both the chemical space and graph space. Especially, our model outperforms

GDSS, in most of the metrics, verifying that our proposed spectral diffusion is not only suitable for generic graph generation but also advisable for molecule designs.

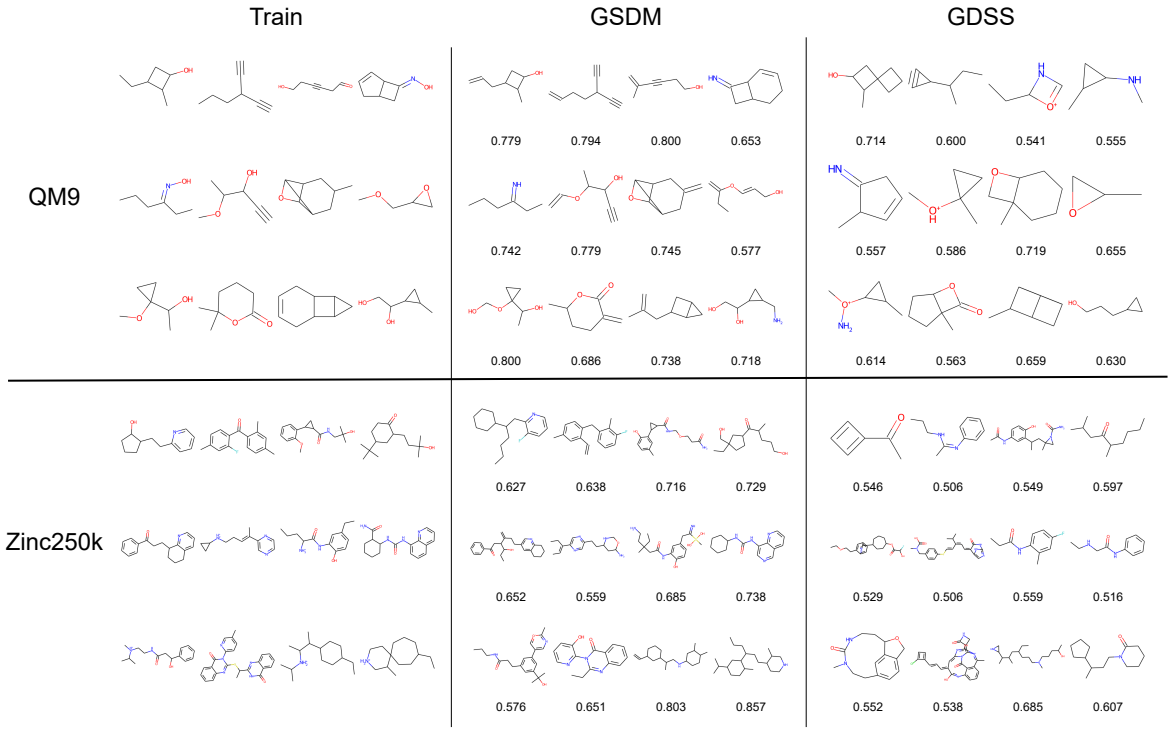


Fig. 5.4: Visualization of molecule generation with maximum Tanimoto similarity of GSDM comparing to GDSS. The left part shows randomly selected molecules from the training set of QM9 and Zinc250k. For each generated molecules, we show the Tanimoto similarity value at the bottom.

Visualization of molecule generations Apart from the visualization of generated graphs in Figure 5.2, visualizations of generated molecules are showed in Figure 5.4. In the figure, we show the generated molecules that are maximally similar to certain training molecules. We compute the molecule similarity using the Tanimoto similarity based on the Morgan fingerprints, which are implemented based on REKit [111]. Higher Tanimoto scores indicate that the model is able to generate more similar molecules as the training set, which reflects the learning capability of the model. As shown in the figure, compared to the GDSS model, GSDM is able to generate molecules that have a more similar distribution as the molecules in the training set.

Time Complexity One of the main advantages of our GSDM compared to other diffusion models (e.g. EDP-GNN and GDSS) is the efficiency in generating molecules. We show the time spent (in seconds) for generating 10,000 molecules in Table 5.2, demonstrating that our GSDM takes a significantly lower time in the inference process compared to EDP-GNN and GDSS. Especially, compared to GDSS which requires $1.06e^3$ and $2.11e^3$ seconds to generate 10,000 molecule graphs according to QM9 and Zinc250k, our model

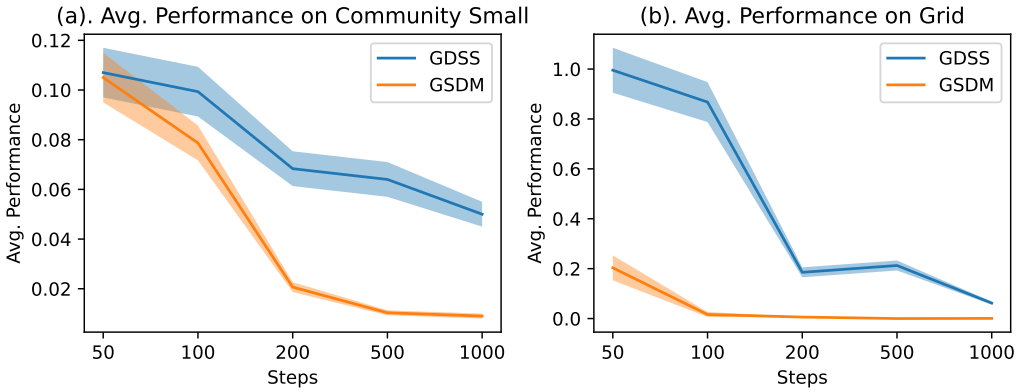


Fig. 5.5: Ablation studies on different diffusion step numbers.

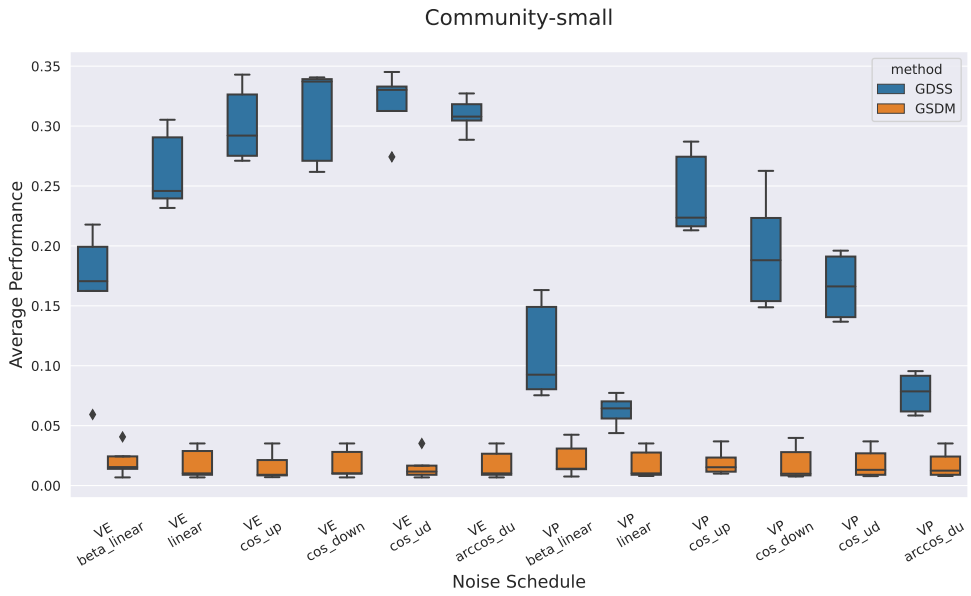
only takes 18.02 and 45.91 seconds, which are **58**× and **46**× faster respectively. This phenomenon verifies our methodology in designing the spectral diffusion that not only the spectral diffusion is more effective than the whole space diffusion, but also more efficient. Such an improvement is crucial for numerous applications such as drug design and material analysis.

Table 5.3: **Ablation study on the α-quantile eigenvalues.** The metrics used are the same as in Table 5.1. The results that **surpass** the baseline methods in Table 5.1 are highlighted in bold (the smaller the better). Avg.% denotes the percentage of average scores achieved compared to using the whole eigenvalues and eigenvectors set.

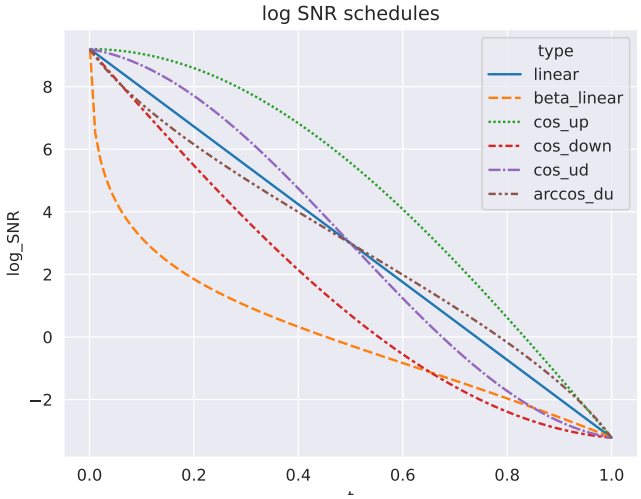
	Community-small					Enzymes					Grid				
	Synthetic, $12 \leq \|V\| \leq 20$					Real, $10 \leq \|V\| \leq 125$					Synthetic, $100 \leq \|V\| \leq 400$				
α	Deg.↓	Clus.↓	Orbit↓	Avg.↓	Avg.%↑	Deg.↓	Clus.↓	Orbit↓	Avg.↓	Avg.%↑	Deg.↓	Clus.↓	Orbit↓	Avg.↓	Avg.%↑
10%	1.010	1.105	0.227	0.781	0.012	0.446	0.608	0.059	0.371	0.029	1.996	**0**	1.013	1.003	0.001
20%	0.450	1.102	0.102	0.551	0.016	0.056	0.220	0.012	0.096	0.395	1.996	**0**	1.013	1.003	0.001
30%	0.085	0.421	0.030	0.179	0.050	**0.011**	**0.093**	**0.012**	**0.039**	0.982	**0.107**	**0**	**0.044**	**0.050**	0.012
40%	**0.039**	**0.056**	**0.006**	**0.034**	0.268	**0.010**	**0.093**	**0.011**	**0.038**	1.000	**0.018**	**0**	**0.001**	**0.006**	0.095
50%	**0.022**	**0.024**	**0.002**	**0.016**	0.563	**0.010**	**0.093**	**0.011**	**0.038**	1.000	**0.007**	**0**	**0.001**	**0.002**	0.225
60%	**0.014**	**0.019**	**0.001**	**0.011**	0.794	**0.010**	**0.093**	**0.011**	**0.038**	1.000	**0.002**	**0**	**0**	**0.001**	1.000
70%	**0.012**	**0.019**	**0.001**	**0.011**	0.844	**0.010**	**0.093**	**0.011**	**0.038**	1.000	**0.002**	**0**	**0**	**0.001**	1.000
80%	**0.011**	**0.016**	**0.001**	**0.009**	0.964	**0.010**	**0.093**	**0.011**	**0.038**	1.000	**0.002**	**0**	**0**	**0.001**	1.000
90%	**0.011**	**0.015**	**0.001**	**0.009**	1.000	**0.010**	**0.093**	**0.011**	**0.038**	1.000	**0.002**	**0**	**0**	**0.001**	1.000
100%	**0.011**	**0.015**	**0.001**	**0.009**	1.000	**0.010**	**0.093**	**0.011**	**0.038**	1.000	**0.002**	**0**	**0**	**0.001**	1.000

5.5.3 Ablation Studies

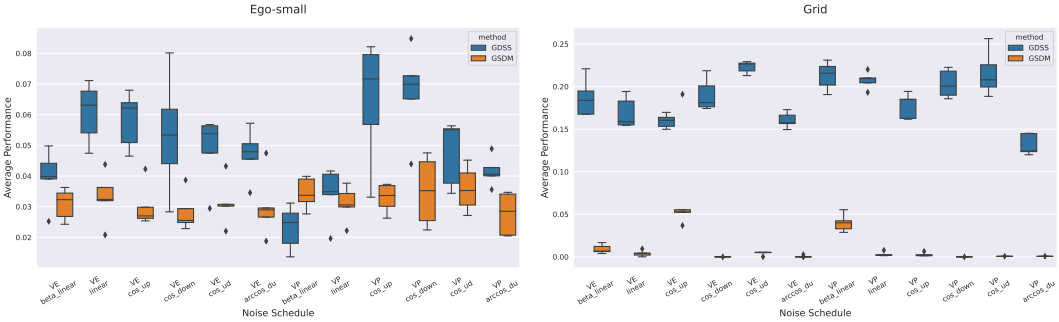
Ablation studies on the number of diffusion steps. To analyze the robustness of our proposed GSDM during the reconstruction procedure (i.e. reverse diffusion process), we compare the performance of our model with GDSS on varying numbers of diffusion steps using the Community small and Grid dataset, which represents small and large generic graphs respectively. The default steps for our model and GDSS during training



(a) Ablation studies on diffusion schedules. The y-axis denotes the average performance score (↓) and each column denotes a diffusion schedule configuration. Each box plot summarizes the results of 5 random trials.



(b) Noise schedules.



(c) Ablation study on the choice of noise schedules.

Fig. 5.6: Ablation studies on diffusion and noise schedules.

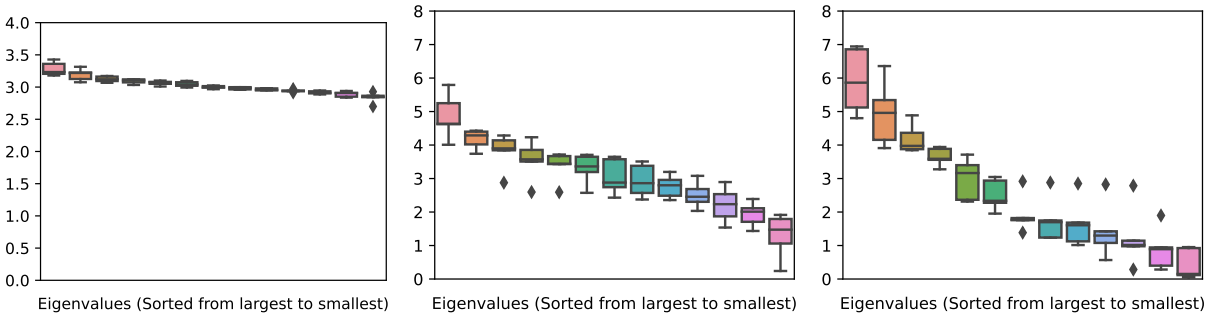
and evaluation are 1,000 steps. Ideally, for diffusion models, with the reduction in step numbers, the performance of the models decreases, while the model of higher robustness still exhibits a higher performance under different numbers of diffusion steps. The results are shown in Figure 5.5. We can observe that GSDM outperforms GDSS in both datasets under different diffusion steps. Specifically, in the Grid dataset, GSDM consistently exhibits significantly higher performance than GDSS. In the community small dataset, although GSDM and GDSS achieve similar performance at 50 steps, GSDM becomes much more accurate when the number of steps is higher than 100.

Ablation studies on the type of diffusion schedules. As mentioned in [152, 188] and [97], the performance of diffusion models highly relies on the diffusion schedule, i.e. the scheduled noise insertion process or discretization of certain types of SDEs, which lies at the heart of both training and inference phases. According to the taxonomy proposed in [188] and [187], mainstream artificial diffusion schedules are categorized to be either Variance Exploding (VE) or Variance Preserving (VP). To improve model robustness against the choice of diffusion schedules, [97] and [152] designed diffusion models with optimizable diffusion schedules. In this subsection, we conduct systematic experiments on several datasets to test the robustness of our proposed GSDM against different diffusion schedules. As illustrated in Figure 5.6a, the state-of-the-art performance of our GSDM is immune to the choice of diffusion schedule and it frees us from tedious hyperparameter searching. When compared with GDSS, GSDM exhibits evidently better average performance with much smaller variance across various configurations.

We further empirically verify the robustness of our proposed GSDM under various types of noise schedules. As shown in Figure 5.6b, we design six representative noise schedules with the same initial and final signal-to-noise ratio (defined in [152,188]), while exhibiting different behavior along the diffusion process. For both VP- and VE-SDE configurations, we train our proposed GSDM with six representative noise schedules and we evaluate the average performance scores. Results on Ego-small and Grid in Figure 5.6c show that GSDM is robust to the choice of diffusion schedules, and it is able to achieve SOTA performance without deliberate noise schedule optimization.

Ablation studies on low-rank approximations of the score-function. As mentioned in Section 5.3.2, one can further accelerate the training and sampling phases, by confining diffusion to the partial spectrum of the graph adjacency matrix. Such variants are coined as α-quantile GSDM in Section 5.3.2. In this experiment, we select the top $[\alpha n]$ eigenvalues to conduct the sampling process with a well-trained GSDM, where α is chosen from $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$. The results are shown in Table 5.3.

Numerical experiments prove that the low-rank α-quantile GSDM retains the state-of-the-art performance among most of the baseline methods. Especially, for the Enzymes dataset, by merely preserving the top-30% eigenvalues and eigenvectors, the 0.3-quantile GSDM can still retain 98.2% of the performance of the full GSDM. The ablation study justifies that our proposed GSDM is capable of capturing essential knowledge that resides on low-dimensional manifolds. It further demonstrates that α-quantile GSDM enables



(a) Synthetic-1 (b) Synthetic-2 (c) Synthetic-3

Fig. 5.7: The eigenvalue distribution of 3 synthetic datasets.

outstanding scalability for large-scale datasets.

Ablation studies on graphs with different eigenvalues distributions. To analyze how GSDM performs under graphs with different eigenvalue distributions, we randomly generate 3 types of synthetic graphs with eigenvalues distributions shown in Figure 5.7: synthetic-1 has evenly distributed eigenvalues, and synthetic-2 and synthetic-2 have moderate and high distinctions in eigenvalues. Due to the predefined eigenvalue distributions, the edge values of the synthetic graphs are not limited to $\{0, 1\}$. Thus instead of using the metrics presented in Section 5.1, we directly compute the MMD of the adjacency matrices between the generated graphs and the test graphs. Corresponding results compared to GDSS are shown in Figure 5.8. We also include an ablation variant of GSDM by using only top-50% quantile eigenvalues. Both GSDM and its ablation variant outperform the GDSS model under this synthetic setting.

5.6 Analysis on Eigenvectors Sampling

In GSDM, we sample eigenvectors from the training set to generate new samples, as elaborated in Section 5.3. In this section, we empirically verify that our GSDM method is able to maintain state-of-the-art performance even when the initial adjacency matrix is randomly sampled.

Specifically, we sample the initial adjacency as $\mathbf{A} \triangleq \widehat{\mathbf{U}}\mathbf{\Lambda}\widehat{\mathbf{U}}^\top$, where $\mathbf{\Lambda}$ is a diagonal Gaussian matrix. We generate $\widehat{\mathbf{U}}$ by first sampling $\mathbf{U}$ from a Gaussian distribution with mean $\bar{\mathbf{U}}$ and covariance matrix $\sigma\mathbf{I}$. Here, $\bar{\mathbf{U}}$ is the empirical mean over the training data, and σ represents the noise magnitude. Finally, we obtain $\widehat{\mathbf{U}}$ by normalizing the ℓ_2 norm of each column of $\mathbf{U}$ to 1. Notice that $\widehat{\mathbf{U}}$ is not necessarily to be orthogonal. We denote this "GSDM with randomly initialized eigenvector" implementation as "GSDM-$\widehat{\mathbf{U}}$".

As shown in Table 5.4, GSDM demonstrates the state-of-the-art performance over Community-small, Enzymes, and Grid datasets, even with a randomly initialized adjacency matrix. Furthermore, the diversity of the generated data is further boosted as the randomness of the initial adjacency matrix increases.

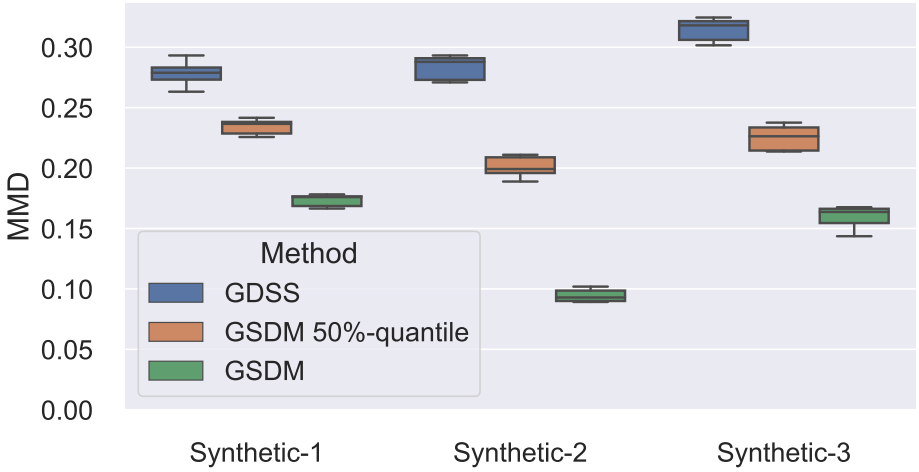


Fig. 5.8: Experiments on synthetic datasets of which the eigenvalues' distributions are shown in Figure 5.7. The y-axis denotes the MMD (↓) of the adjacency matrices between the generated graphs to the test graphs

Table 5.4: **Generation results on the generic graph datasets for GSDM, GSDM-$\hat{\mathbf{U}}$, and GDSS.** We report the MMD distances between the test datasets and generated graphs. The best results are highlighted in bold (the smaller the better).

	Community-small				Enzymes				Grid			
	Synthetic, $12 \leq \|V\| \leq 20$				Real, $10 \leq \|V\| \leq 125$				Synthetic, $100 \leq \|V\| \leq 400$			
	Deg.↓	Clus.↓	Orbit↓	Avg.↓	Deg.↓	Clus.↓	Orbit↓	Avg.↓	Deg.↓	Clus.↓	Orbit↓	Avg.↓
GDSS [91]	0.045	0.086	0.007	0.046	0.026	0.102	**0.009**	0.046	0.111	0.005	0.070	0.062
GSDM	**0.011**	**0.015**	**0.001**	**0.009**	**0.013**	0.088	**0.009**	**0.037**	**0.002**	**0.0**	**0.0**	**0.0007**
GSDM-$\hat{\mathbf{U}}$	0.032	0.073	0.004	0.036	0.023	0.096	**0.009**	0.042	0.002	0.0	0.0	0.0007

In summary, our GSDM is robust to the choice of initial adjacency matrix, and the state-of-the-art performance of GSDM does not rely on empirically sampling the eigenvector matrix from training data. We can increase the noise magnitude of $\widehat{\mathbf{U}}$ to generate more diverse samples.

We further provide a theoretical analysis of the effects of eigenvector sampling on GSDM. While Proposition 2 establishes the reconstruction bound primarily based on empirical eigenvector sampling, we demonstrate that GSDM with randomly initialized eigenvectors (referred to as GSDM-$\widehat{\mathbf{U}}$) retains all the theoretical benefits of GSDM, with only minor revisions. In Proposition 5.4, we prove that GSDM-$\widehat{\mathbf{U}}$ continues to possess a reconstruction bound of order $O(n\exp(n))$, which is still considerably tighter than the $O(n^2\exp(n^2))$ bound of GDSS. When the ground truth distribution is equal to the distribution of $\widehat{\mathbf{U}}$, the new bound reduces to the one presented in Proposition 2.

Proposition 5.4 (Reconstruction bound for GSDM with Gaussian initialized eigen-

vectors.). *Following the notations and conditions in Proposition 2, suppose the adjacency matrix of GSDM is constructed by* $\widetilde{\mathbf{A}}_0^{\text{spec}} \triangleq \widehat{\mathbf{U}}\boldsymbol{\Lambda}_0^{\text{spec}}\widehat{\mathbf{U}}^\top$*, where* $\widehat{\mathbf{U}}$ *is sampled from* $N(\mathbb{E}\mathbf{U}, \sigma\mathbf{I})$ *and then its columns are normalized to* 1*. Here,* $\mathbb{E}\mathbf{U}$ *denotes the expectation of the eigenvectors of the adjacency matrix, and* $\sigma > 0$ *is a predefined constant. Then, the expected reconstruction error is bounded by*

$$\begin{aligned}
&\mathbb{E}\|\mathbf{A}_0 - \widetilde{\mathbf{A}}_0^{\text{spec}}\|_{2,2}^2 \\
&\leqslant CW_2((\widehat{\mathbf{U}}), (\mathbf{U}_0))^2 \left(\mathbb{E}\|\mathbf{A}_0\|_{2,2} + B(n, M, L, \boldsymbol{\phi}, \sigma.)\right) \\
&\quad + CB(n, M, L, \boldsymbol{\phi}, \sigma.),
\end{aligned} \tag{5.23}$$

where $B(n, M, L, \boldsymbol{\phi}, \sigma.)$ *is the reconstruction bound for GSDM in Proposition 2;* C *is an absolute constant;* $W_2(\cdot, \cdot)$ *is the Wasserstein 2-distance between two probability measures.*

Proof: [Proof of Proposition 5.4] Based on the proof presented in Proposition 2 of the main text, only a few additional revisions are required. In fact, we have

$$\begin{aligned}
&\mathbb{E}\|\mathbf{A}_0 - \widetilde{\mathbf{A}}_0^{\text{spec}}\|_{2,2}^2 \\
&= \mathbb{E}\|\mathbf{U}\boldsymbol{\Lambda}_0\mathbf{U}^\top - \widehat{\mathbf{U}}\widetilde{\boldsymbol{\Lambda}}_0^{\text{spec}}\widehat{\mathbf{U}}^\top\|_{2,2}^2 && (5.24)\\
&\leqslant 2\big(\mathbb{E}\|\mathbf{U}(\boldsymbol{\Lambda}_0 - \widetilde{\boldsymbol{\Lambda}}_0^{\text{spec}})\mathbf{U}^\top\|_{2,2}^2 \\
&\quad + \mathbb{E}\|\mathbf{U}\widetilde{\boldsymbol{\Lambda}}_0^{\text{spec}}\mathbf{U}^\top - \widehat{\mathbf{U}}\widetilde{\boldsymbol{\Lambda}}_0^{\text{spec}}\widehat{\mathbf{U}}^\top\|_{2,2}^2\big) && (5.25)\\
&\leqslant 2\big(B(n, M, L, \boldsymbol{\phi}, \sigma.) \\
&\quad + \mathbb{E}\|\widehat{\mathbf{U}}\widetilde{\boldsymbol{\Lambda}}_0^{\text{spec}}\widehat{\mathbf{U}}^\top - \widehat{\mathbf{U}}\widetilde{\boldsymbol{\Lambda}}_0^{\text{spec}}\widehat{\mathbf{U}}^\top\|_{2,2}^2\big). && (5.26)
\end{aligned}$$

Notice that

$$\begin{aligned}
&\|\mathbf{U}\widetilde{\boldsymbol{\Lambda}}_0^{\text{spec}}\mathbf{U}^\top - \widehat{\mathbf{U}}\widetilde{\boldsymbol{\Lambda}}_0^{\text{spec}}\widehat{\mathbf{U}}^\top\|_{2,2} \\
&\leqslant \|\mathbf{U}\widetilde{\boldsymbol{\Lambda}}_0^{\text{spec}}(\mathbf{U} - \widehat{\mathbf{U}})^\top\|_{2,2} + \|(\mathbf{U} - \widehat{\mathbf{U}})\widetilde{\boldsymbol{\Lambda}}_0^{\text{spec}}\widehat{\mathbf{U}}^\top\|_{2,2} \\
&\leqslant \|\mathbf{U} - \widehat{\mathbf{U}}\|_2(\|\mathbf{U}\|_2 + \|\widehat{\mathbf{U}}\|_2)\|\widetilde{\boldsymbol{\Lambda}}_0^{\text{spec}}\|_{2,2} && (5.27)\\
&\leqslant \|\mathbf{U} - \widehat{\mathbf{U}}\|_2(1 + \|\widehat{\mathbf{U}}\|_2)\|\widetilde{\boldsymbol{\Lambda}}_0^{\text{spec}}\|_{2,2}. && (5.28)
\end{aligned}$$

By using the conclusion in Proposition 2, we have

$$\begin{aligned}
&\mathbb{E}\|\mathbf{A}_0 - \widetilde{\mathbf{A}}_0^{\text{spec}}\|_{2,2}^2 \\
&\leqslant \mathbb{E}\|\mathbf{U} - \widehat{\mathbf{U}}\|_2^2(1 + \|\widehat{\mathbf{U}}\|_2)^2\mathbb{E}\|\widetilde{\boldsymbol{\Lambda}}_0^{\text{spec}}\|_{2,2}^2 && (5.29)\\
&\leqslant C\mathbb{E}\|\mathbf{U} - \widehat{\mathbf{U}}\|_2^2 B(n, M, L, \boldsymbol{\phi}, \sigma.) && (5.30)\\
&\leqslant CW_2((\widehat{\mathbf{U}}), (\mathbf{U}_0))^2 B(n, M, L, \boldsymbol{\phi}, \sigma.), && (5.31)
\end{aligned}$$

which completes the proof.

5.7 Analysis on Eigenvector Diffusion

In this section, we discuss the reason why we do not incorporate eigenvector diffusion into our GSDM model. We show that the eigenvector diffusion can degrade the performance GSDM.

In theory, there are two ways to perform diffusion on the eigenvectors: one can either perform the diffusion on the full space $\mathbb{R}^{n\times n}$ or on the Stiefel manifold $V_n(\mathbb{R}^n) \triangleq \{\mathbf{U} \in \mathbb{R} \mid \mathbf{U}^\top\mathbf{U} = \mathbf{I}\}$.

Diffusion on the full space $\mathbb{R}^{n\times n}$. In Proposition 5.5, we prove that the reconstruction error bound of the eigenvector diffusion on the full space (e.g. $\mathbb{R}^n$) is at the order of $\mathcal{O}(n^3\exp(n^2+n))$, which is substantially larger than the order of GSDM $\mathcal{O}(n\exp(n))$. This shows that learning the eigenvector via full space diffusion is infeasible, especially when the graph size n is extremely large. As the eigenvectors reside on the Stiefel manifold, the diffusion SDE in the ambient space introduces irreducible noise to the eigenvectors, making it challenging to recover them from the reversed SDE. This conclusion aligns with our previous experiment in the first rebuttal, where we observed that GSDM outperforms the eigenvector diffusion counterpart in practice.

Proposition 5.5 (Reconstruction bound of eigenvector diffusion.)**.** *Following the notations and assumptions in Proposition 2, suppose we introduce an additional score neural network $s_{\boldsymbol{\psi}}(\cdot)$ to learn to reverse the diffusion SDE on the eigenvectors, i.e.*

$$\dot{\mathbf{\Lambda}}_t = \mathbf{f}^{\Lambda}(\bar{\mathbf{\Lambda}}_t, t)\dot{t} + \sigma_{\Lambda,t}\dot{B}_t^{\Lambda}, \tag{5.32}$$

$$\dot{\mathbf{U}}_t = \mathbf{f}^{U}(\bar{\mathbf{U}}_t, t)\dot{t} + \sigma_{U,t}\dot{B}_t^{U}. \tag{5.33}$$

By reversing the SDE $(\mathbf{U}_t, \mathbf{\Lambda}_t)_{t\in[0,1]}$, the adjacency matrix is generated by $\hat{\mathbf{A}}_t^{\text{vec}} \triangleq \bar{\mathbf{U}}_t\bar{\mathbf{\Lambda}}_t\bar{\mathbf{U}}_t^\top$. If we further assume that $\|s_{\boldsymbol{\psi}}(\cdot,t)\|_{\text{lip}} = \mathcal{O}(\mathbb{E}_{|\mathbf{U}_0}\|\nabla_{\mathbf{U}}\log p_{t|0}(\cdot,t)\|_{\text{lip}})$, the expected reconstruction error is bounded by

$$\begin{aligned}
&\mathbb{E}\|\mathbf{A}_0 - \hat{\mathbf{A}}_0^{\text{vec}}\|^2 \\
&\leqslant 2(CB(n^2, M, L, \boldsymbol{\psi}, \sigma.)(B(n, M, L, \boldsymbol{\phi}, \sigma.) + \mathbb{E}\|\mathbf{A}_0\|_{2,2}^2) \\
&\quad + B(n, M, L, \boldsymbol{\phi}, \sigma.)),
\end{aligned} \tag{5.34}$$

where $B(n, M, L, \boldsymbol{\phi}, \sigma.) = \mathcal{O}(n\exp(n))$ is the reconstruction bound function for GSDM in Proposition 2, and C is an absolute constant.

Proof: [Proof of Proposition 5.5] Using the proof of Proposition 2 of the main text, we

have

$$\begin{aligned}
&\mathbb{E}\|\mathbf{A}_0 - \hat{\mathbf{A}}_0^{\text{vec}}\|_{2,2}^2 \\
&= \mathbb{E}\|\mathbf{U}_0\mathbf{\Lambda}_0\mathbf{U}_0^\top - \bar{\mathbf{U}}_0\bar{\mathbf{\Lambda}}_0^{\text{spec}}\bar{\mathbf{U}}_0^\top\|_{2,2}^2 && (5.35)\\
&\leqslant 2\big(\mathbb{E}\|\mathbf{U}_0(\mathbf{\Lambda}_0 - \bar{\mathbf{\Lambda}}_0^{\text{spec}})\mathbf{U}_0^\top\|_{2,2}^2 \\
&\quad + \mathbb{E}\|\mathbf{U}_0\bar{\mathbf{\Lambda}}_0^{\text{spec}}\mathbf{U}_0^\top - \bar{\mathbf{U}}_0\bar{\mathbf{\Lambda}}_0^{\text{spec}}\bar{\mathbf{U}}_0^\top\|_{2,2}^2\big) && (5.36)\\
&\leqslant 2\big(B(n, M, L, \boldsymbol{\phi}, \sigma_\cdot) \\
&\quad + \mathbb{E}\|\mathbf{U}_0\bar{\mathbf{\Lambda}}_0\mathbf{U}_0^\top - \bar{\mathbf{U}}_0\bar{\mathbf{\Lambda}}_0^{\text{spec}}\bar{\mathbf{U}}_0^\top\|_{2,2}^2\big). && (5.37)
\end{aligned}$$

By applying Proposition 1 to the reversed diffusion process of $(\mathbf{U}_t)_{t\in[0,1]}$, we have

$$\begin{aligned}
&\|\mathbf{U}_0\bar{\mathbf{\Lambda}}_0^{\text{spec}}\mathbf{U}_0^\top - \bar{\mathbf{U}}_0\bar{\mathbf{\Lambda}}_0^{\text{spec}}\bar{\mathbf{U}}_0^\top\|_{2,2} \\
&\leqslant \|\mathbf{U}_0\bar{\mathbf{\Lambda}}_0^{\text{spec}}(\mathbf{U}_0 - \bar{\mathbf{U}}_0)\|_{2,2} + \|\bar{\mathbf{U}}_0\bar{\mathbf{\Lambda}}_0^{\text{spec}}(\mathbf{U}_0 - \bar{\mathbf{U}}_0)\|_{2,2} \\
&\leqslant \|\mathbf{U}_0 - \bar{\mathbf{U}}_0\|_2(\|\mathbf{U}_0\|_2 + \|\bar{\mathbf{U}}_0\|_2)\|\mathbf{\Lambda}_0^{\text{spec}}\|_{2,2} && (5.38)\\
&\leqslant C\|\mathbf{U}_0 - \bar{\mathbf{U}}_0\|_2\|\mathbf{\Lambda}_0^{\text{spec}}\|_{2,2}, && (5.39)
\end{aligned}$$

and

$$\begin{aligned}
&\mathbb{E}\|\mathbf{U}_0\bar{\mathbf{\Lambda}}_0^{\text{spec}}\mathbf{U}_0^\top - \bar{\mathbf{U}}_0\bar{\mathbf{\Lambda}}_0^{\text{spec}}\bar{\mathbf{U}}_0^\top\|_{2,2}^2 \\
&\quad \leqslant CB(n^2, M, L, \boldsymbol{\psi}, \sigma_\cdot)\mathbb{E}\|\mathbf{\Lambda}_0^{\text{spec}}\|_{2,2}^2, && (5.40)
\end{aligned}$$

and the proof is completed by

$$\begin{aligned}
&\mathbb{E}\|\mathbf{A}_0 - \hat{\mathbf{A}}_0^{\text{vec}}\|_{2,2}^2 \\
&\leqslant 2\left(B(n, M, L, \boldsymbol{\phi}, \sigma_\cdot) + \mathbb{E}\|\mathbf{U}_0\bar{\mathbf{\Lambda}}_0\mathbf{U}_0^\top - \bar{\mathbf{U}}_0\bar{\mathbf{\Lambda}}_0^{\text{spec}}\bar{\mathbf{U}}_0^\top\|_{2,2}^2\right) \\
&\leqslant 2(B(n, M, L, \boldsymbol{\phi}, \sigma_\cdot) + CB(n^2, M, L, \boldsymbol{\psi}, \sigma_\cdot)\mathbb{E}\|\mathbf{\Lambda}_0^{\text{spec}}\|_{2,2}^2) \\
&\leqslant 2(CB(n^2, M, L, \boldsymbol{\psi}, \sigma_\cdot)(B(n, M, L, \boldsymbol{\phi}, \sigma_\cdot) + \mathbb{E}\|\mathbf{A}_0\|_{2,2}^2) \\
&\quad + B(n, M, L, \boldsymbol{\phi}, \sigma_\cdot)). && (5.41)
\end{aligned}$$

Diffusion on the Stiefel Manifold $V_n(\mathbb{R}^n)$. A more elegant way to perform eigenvector diffusion is to leverage the diffusion SDE on the Stiefel manifold. In fact, the study of diffusion on the manifold is indeed a distinct topic, falling outside the scope of this chapter.

The main challenge of leveraging manifold SDE to learn eigenvectors is the lack of global parameterization of the manifold, which makes the numerical computation not straightforward to generate the Wiener process on the Stiefel manifold. As proposed in [243], the forward Stiefel manifold SDE can be sampled via the Intermittent Diminishing Diffusion on Stiefel Manifold (IDDM). However, at each diffusion step, the IDDM algorithm involves an inner loop to calculate the local gradient on the manifold, which is computationally intensive.

Even if we can efficiently sample the Wiener SDE on the manifold, the reversed SDE lacks a closed-form density function, which poses challenges in efficiently training a diffusion model with the standard score-matching objective. As shown in [29], the Riemannian score-matching objective needs to be implicitly approximated by integrating the logarithm of the transition probability of the manifold SDE. This process will introduce significant overhead in the training process.

We would like to emphasize that our primary goal is to achieve better acceleration and encourage our model to learn low-rank distributions. Therefore, we opted not to incorporate the dense and computationally costly manifold SDE into our GSDM. In Table 5.7, we have empirically shown that the eigenvector diffusion harms the performance of GSDM. Overall, our GSDM is efficient yet effective in generating high-quality graph data.

Table 5.5: **Diversity results on the generic graph datasets.** We report the *precision*, *recall*, *coverage* and *MMD-RBF* on Community-small, Enzymes, and Grid datasets. The best results are highlighted in bold.

	Community-small				Enzymes				Grid			
	Synthetic, $12 \leq \|V\| \leq 20$				Real, $10 \leq \|V\| \leq 125$				Synthetic, $100 \leq \|V\| \leq 400$			
	Re.↑	Cov.↑	MMD-RBF.↓	Nov. ↑	Re.↑	Cov.↑	MMD-RBF.↓	Nov. ↑	Re.↑	Cov.↑	MMD-RBF.↓	Nov. ↑
GDSS [91]	0.350	0.550	0.166	**0.980**	0.921	0.968	0.148	**1**	0.550	**1**	0.201	0.950
GSDM-Û	0.932	0.950	0.021	**0.980**	**0.968**	**0.995**	0.081	**1**	**1**	**1**	0.043	0.950
GSDM	**0.961**	**1**	**0.017**	0.950	0.859	0.890	**0.021**	0.930	**1**	**1**	0.043	0.830

5.8 Evaluation of Generation Diversity

In this section, we empirically prove that GSDM is able to generate diverse, plausible, and novel graph data. We first introduce two diversity metrics for general graph generation and molecule generation. On top of that, our results demonstrate that GSDM does not simply replicate samples from the training set.

Diversity for generic graph generation. According to [110], for graph data from general domains, a good generative model that generates diverse samples should be able to avoid both mode collapse and model dropping. Specifically, suppose the real and generated distributions are P_r and P_g, mode collapse means P_g exhibits repeating patterns that lack diversity, while mode dropping means P_g fails to capture all modes shown in P_r. Hence, [245] measures the diversity of a generative model with quantitative metrics that are highly correlated to mode collapse and mode dropping, including **recall**, **coverage**, and **MMD-RBF**. Their definitions are detailed below.

- **Recall (Re.)**: the percentage of real samples that fall within the manifold of generated samples.

Table 5.6: **Molecule generation diversity measure on the QM9 and ZINC250k datasets.** Results are the means of three different runs, and the best results are highlighted in bold.

		QM9		ZINC250k	
	Method	Uniqueness (%)↑	Novelty (%)↑	Uniqueness (%)↑	Novelty (%)↑
Autoreg.	GraphNVP	99.2	58.2	94.8	100
	GRF	66.0	58.6	53.7	100
	GraphAF	94.51	88.83	99.1	100
One-shot	MoFlow	99.20	98.03	99.99	100
	GDSS	97.87	87.13	99.79	100
	GSDM	98.66	92.15	99.99	100

Table 5.7: **Generation results on the generic graph datasets for GSDM, GSDM-$\mathbf{U}_\theta$, and GDSS.** We report the MMD distances between the test datasets and generated graphs. The best results are highlighted in bold (the smaller the better).

	Community-small				Enzymes				Grid			
	Synthetic, $12 \leq \lvert V\rvert \leq 20$				Real, $10 \leq \lvert V\rvert \leq 125$				Synthetic, $100 \leq \lvert V\rvert \leq 400$			
	Deg.↓	Clus.↓	Orbit↓	Avg.↓	Deg.↓	Clus.↓	Orbit↓	Avg.↓	Deg.↓	Clus.↓	Orbit↓	Avg.↓
GDSS [91]	0.045	0.086	0.007	0.046	0.026	0.102	**0.009**	0.046	0.111	0.005	0.070	0.062
GSDM	**0.011**	**0.015**	**0.001**	**0.009**	**0.013**	**0.088**	**0.009**	**0.037**	**0.002**	**0.0**	**0.0**	**0.0007**
GSDM-$\mathbf{U}_\theta$	0.042	0.079	0.007	0.043	0.026	0.099	**0.009**	0.045	0.059	0.002	0.032	0.031

- **Coverage (Cov.)**: the percentage of real hypersphere that contains a generated sample.
- **MMD-RBF**: Compute MMD between the empirical distribution of real the and generated samples with the RBF kernel: $k\left(\mathbf{x}_i, \mathbf{x}_j\right) = \exp\left(-d\left(\mathbf{x}_i, \mathbf{x}_j\right)/2\sigma^2\right)$

In addition, we propose a "**novelty (Nov.)**" metric to measure the novelty of generated samples. In particular, the "novelty" score is defined as the percentage of the generated samples that do not have the same adjacency matrix as anyone in the training data. The "novelty" score directly reflects whether the generative model is simply recapitulating the training set.

In Table 5.5, we benchmark the diversity of GDSS, GSDM, and GSDM-$\hat{\mathbf{U}}$ with the aforementioned diversity metrics on Community-Small, Enzyme, and Grid datasets. **Recall** and **coverage** have strong positive rank correlations with the diversity so that high **recall** and **coverage** indicate low mode collapse and low mode dropping. **MMD-RBF** also has a strong positive rank correlation with the diversity, and a lower MMD-RBF value indicates higher diversity. These results imply that GSDM and GSDM-$\hat{\mathbf{U}}$ have a better capability to avoid mode dropping and mode collapse than GDSS.

We evaluate the novelty score of GSDM to determine whether it replicated samples

from the training set. The results revealed that GSDM is capable of producing novel graphs with high scores of 0.95, 0.93, and 0.83 in three different datasets, respectively. Furthermore, randomly sampling $\hat{\mathbf{U}}$ increases the novelty score of GSDM to 0.98, 1, and 0.95 in the respective datasets, effectively preventing GSDM from generating samples that closely resemble those in the training set.

Diversity for molecule generation. According to [183, 245], the most widely used diversity measures for molecule generation are the **novelty** and **uniqueness** scores.

- **Uniqueness**: the percentage of unique and valid molecules among the generated molecules.
- **Novelty**: the percentage of validly generated molecules that are not in the training dataset.

A high value of uniqueness indicates a reduced risk of mode collapse, which implies the model is able to generate various diverse molecules. When a model exhibits high novelty, it means that it has a greater tendency to generate molecules that are not present in the training set, thus demonstrating its ability to generate new and previously unseen molecules. We follow the same setting in [245] to evaluate the novelty of our GSDM in molecule generation on $10,000$ generated samples.

As shown in Table 5.6, for the Zinc250k dataset, our GSDM achieves high uniqueness and novelty score with 99.99% and 100% respectively, demonstrating that our GSDM is competent to generate diverse and valid novel molecules, rather than simply replicates samples from the training set.

5.9 Conclusion and Future Work

In this chapter, we present a novel graph diffusion model named GSDM, which consists of diffusion methods on both graph node features and topologies through SDEs. Specifically, for graph topology diffusion, we design a novel spectral diffusion model in GSDM, to improve the accuracy of predicting score functions, and avoid the pitfalls of the conventional diffusion processes on graphs. Furthermore, we provide theoretical analysis to justify the advantages in effectiveness and efficiency of GSDM compared to the standard graph diffusion. To validate our proposed model, we conducted extensive experiments showing that our proposed GSDM outperforms the state-of-the-art baseline methods on both generic graph generation and molecule generation with significantly higher processing speed.

Note that in GSDM, we propose to perform a diffusion process on the eigenvalues of the graph adjacency matrix. We also present some ideas about performing diffusion on the eigenvectors of the graph adjacency matrix and discuss the potential issues of the eigenvector-diffusion paradigms in Section 5.7, and provide some preliminary empirical studies. In the future, we will explore the possibility of developing an efficient algorithm to perform diffusion on eigenvectors for graph generation.

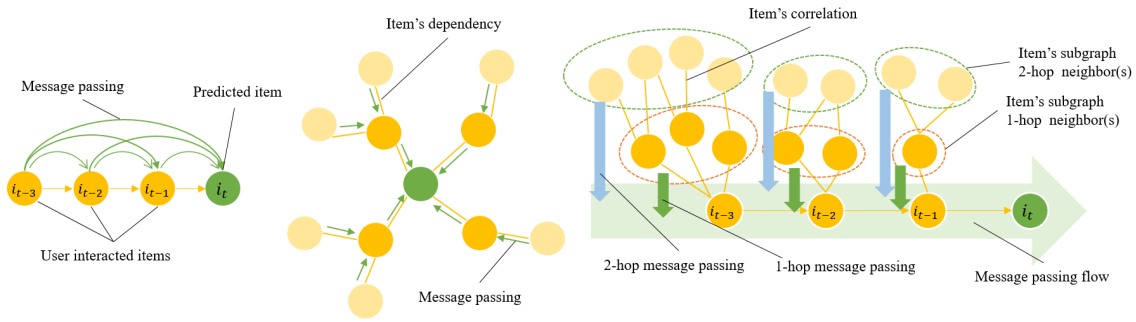
Chapter 6

Breaking the Limit of Message Passing in Sequential Recommendations: A Multi-view GNN-Transformer Architecture

6.1 Overview

Recommendation models are one of the major applications of Graph neural networks. In this chapter, we focus on the application of designing GNN models for recommender systems. Extracting user preferences from the user and item correlation graphs has been proven to be useful in improving the capability of predicting user preferences. NGCF [220] extracts user and item representation from the user-item graph, to achieve accurate collaborative filtering (CF). LightGCN [74] achieves state-of-the-art graph-based CF performance with higher efficiency than NGCF. Graph-based CF can generally outperform MLP-based CF models such as BPR-MF [169] and NeuMF [73]. However, introducing graph neural networks (GNN) to transformer models is intrinsically difficult, as they require completely different message passing strategies. Typical models of GNNs such as GCN [98], GAT [208], NGCF [220] or LightGCN [74], need to process the entire graph adjacency or Laplacian matrix, to obtain the localized representations for each node, with time complexity proportional to the total number of edges; while transformers only require traces of user behaviors. Thus, stacking GNNs and transformers is not efficient: in each batch, GNNs compute the localized representation for each node, but only a small amount of node features (along the user behavior trace) will be passed to the transformers. Such a process can cause great computational waste, especially for large graphs such as user-item graphs from real-world recommender systems.

We show the common message passing strategies of sequential recommendations as well as our proposed message passing architecture in Figure 6.1. LSTM-based model or transformer-based model such as [92, 192] usually follows diagram (a) which passes



(a) Sequential message passing (b) Message passing in graphs (c) Our proposed message passing architecture

Fig. 6.1: Message passing in sequential recommendations.

messages following the user behavior sequence. Graph-based sequential recommendation methods [158, 166, 219, 222, 226, 242] usually construct user session graphs, and extract information along the edges of the graphs with GNN networks, as shown in Fig 6.1 (b).

In this Chapter, we propose a novel message passing architecture (shown in Fig 6.1 (c)) which not only pass messages between correlated items but also along the user behavior sequences, while solving the inefficiency problem for combining the advantages of GNNs with transformers. First, we define the item correlation item i to j as a probability of the user interacting item j given the user have interacted with item i. Thus, we construct an item correlation graph, where each node represents an item and each edge represents the correlation of both items. We extract representation from 1 to k hop sub-graphs of each node with our proposed hierarchical graph aggregation model, and pass the sub-graph representation to a transformer-based model. Meanwhile, we can form 1 to k hop views of each sub-graph of items: the larger sub-graph, the more item-correlation information is preserved; and the smaller sub-graph, the more chronological relationship of the current user behaviors is enhanced. As the graph model incorporates general item relationships instead of the user-specific chronological representations by the sequential model, compared to a single view, the multi-view architecture can form comprehensive representations of the user behaviors. Finally, we aggregate the multiple views and obtain the final preference representations of the user. In addition, corresponding to our hierarchical graph aggregation, we design the Dirichlet sampling method to improve the robustness and generalization capability of the graph aggregation model.

Moreover, to show the performance of our model, we conduct extensive experiments on four recommendation datasets compared with the state-of-the-art sequential recommendation methods. The results show that our model outperforms the state-of-the-art recommendation models.

Our main contributions are summarized as follows.

- We propose a hierarchical graph aggregation model to support efficient graph aggregation operations on graphs, and analyze its running complexity, comparing with commonly used graph aggregation methods.

- We apply the proposed hierarchical graph aggregation model to build a novel sequential recommendation framework, which has a better capability of user behavior modeling than existing sequential recommendation models.
- We also design a multi-view architecture and propose the Dirichlet sampling method to improve the performance and robustness of the sequential recommendation model. We show the findings in the ablation studies.
- We conduct extensive experiments on five real datasets with comparisons to state-of-the-art sequential recommendation methods. The experimental results show that the proposed model obtains superior performance than baseline methods.

6.2 Methodology

6.2.1 Preliminaries

In this work, we denote the set of users by $\mathcal{U}$ and the set of items by $\mathcal{I}$. For each user u, we denote the sequence of the user's interacted items by $\mathcal{I}_u = [i_1^u, i_2^u, \cdots, i_{n_u}^u]$, where the items are sorted in chronological order based on the interaction timestamps, and n_u denotes the length of the item sequence $\mathcal{I}_u$. Moreover, we denote the set of all the observed user behavior sequences by $\mathcal{D}_s$. The main objective of a sequential recommendation model is to first predict the probability that a user u interacts with an item i, given the list of the user's historical items $\mathcal{I}_u$. Then, the candidate items are ranked based on the predicted dependency probabilities in descending order, and the N top-ranked items are recommended to the user. Table 6.1 summarizes the most used notations in this Chapter.

With all the observed user behavior sequences $\mathcal{D}_s$, we first use a directed dependency graph $\mathcal{G}$ to describe the sequential dependency relationship between two items in the user behavior sequences. Specifically, we use $\mathbf{T}_{i,j}$ to denote item j's dependency on item i, and define it as follows,

$$\mathbf{T}_{i,j} = \frac{\sum_{u \in \mathcal{U}} \mathbb{I}[t_{u,i} < t_{u,j}]}{\sum_{u \in \mathcal{U}} \sum_{k \in \mathcal{I}} \mathbb{I}[t_{u,i} < t_{u,k}]}, \tag{6.1}$$

where $t_{u,i}$ is the timestamp when user u interacts with item i, and $\mathbb{I}(\cdot) = 1$ if the condition is true, and 0 otherwise. In Eq. (6.1), a higher value of $\mathbf{T}_{i,j}$ indicates a higher chance that a user would like to interact with the item j, given that she has interacted with the item i. Then, if $\mathbf{T}_{i,j} > 0$, we build an edge from item i to item j in $\mathcal{G}$. It is worth noting that $\mathbf{T}$ serves as the transition probability of any user clicking item j after she has clicked item i, and the item dependency (transition) matrix $\mathbf{T} \in \mathbb{R}^{n \times n}$ is an asymmetric row-stochastic matrix, the sum of each row is 1, where n is the number of items.

We further define the k-hop item transition matrix $\mathbf{T}^{(k)} = \prod_{i=1}^{k} \mathbf{T}$, and we denote the element in the i-th row, j-th column of $\mathbf{T}^{(k)}$ as $\mathbf{T}_{i,j}^{(k)}$, which also means the transition probability of any user to click item j after he/she has clicked item i k-th steps ago. When

Table 6.1: List of notations.

Notations	Description
$\mathcal{U}$, $\mathcal{I}$	The set of users and items
$\mathcal{I}_u$	The item sequence interacted by u
$\mathcal{D}_s$	The set of observed user behavior sequences
$\mathcal{G}$	The item-item correlation graph
$\mathbb{E}[\cdot]$	The expectation
L_u	The sequence length of user u
L	The maximum input length to the model
d	The dimension of latent representations
$\tau_{u,i}^{(k)}$	The transition probability from item i to item j at k steps
$r_{u,i}$	The interaction score of the $\{u, i\}$ pair
$\mathbf{u}_k$	The k-th view of user representations
θ	The neural network parameters
θ_k	The neural network parameters for the k-th view
$\mathbf{T}^{(k)}$	The k-hop item dependency (transition) matrix
$\mathbf{T}_{i,j}^{(k)}$	The k-hop item dependency score from item i to item j
$\mathcal{N}_i^{(k)}$	The k-hop neighbours of the item i
$\mathbf{e}_i$	The embedding of node i
$\{\alpha_k\}_{k=1}^K$	The set of hyper-parameters for the Dirichlet distribution
$\mathcal{D}(x\|\alpha)$	The Dirichlet distribution for random variable x

k increases, the number of k-hop neighbours increases exponentially, and we may obtain a large amount of k-hop neighbours with low transition probability $\mathbf{T}_{i,j}^{(k)}$. Small transition probability indicates low dependencies, thus such neighbours are not informative enough for message passing, and they can be dropped during training to increase the model processing speed. To this end, in this work, we propose to choose the top h neighbours for each hop to increase the processing speed, while capturing enough information from the global context. Thus, to compute $\mathbf{T}^{(k)}$, we calculate by $\mathbf{T}^{(k)} = \sigma(\mathbf{T}^{(k-1)})\sigma(\mathbf{T})$, where $\sigma(\cdot)$ denotes the filter method to select the top h elements for each row.

Note that the k-hop transition matrix can be computed in the data prepossessing stage. As $\sigma(\mathbf{T}^{(k-1)})\sigma(\mathbf{T})$ is a sparse matrix multiplication, the complexity for each matrix multiplication can be bounded by $\min(O(n^{2+o(1)}), O(n^{2.38}))$ [244].

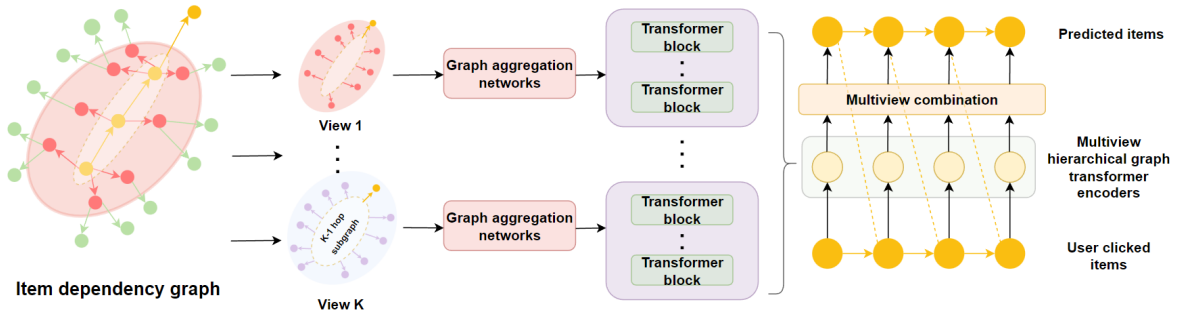


Fig. 6.2: The architecture of the proposed framework. The input contains the user-clicked item sequence together with the sub-graphs of the items from the item dependency graph. We form multiple views for the input item sequence, and each view is passed through the hierarchical graph aggregation networks followed by the transformer encoders. Finally, we combine the representations of the user preference from multiple views to predict the user's next preferred item.

6.2.2 Overall Structure

Figure 6.2 shows the overall architecture of the proposed sequential recommendation framework. For a given item sequence $\mathcal{I}_u$, rather than directly applying sequence encoding methods to obtain the sequence representation, we first extract sub-graphs from $\mathcal{G}$ to augment the sequence $\mathcal{I}_u$. For each item $i \in \mathcal{I}_u$, we extract the sub-graph of $\mathcal{G}$ containing all the 1 to K hop neighbors of i, and then generate different "views" of item i's neighborhood. The sub-graph is also the ego-graph with the node i as the central node, and each view of the ego-graph consists of the nodes which have the same distance to the central node. Specifically, the k-th view only contains the k-hop neighbours of item i.

Figure 6.3 shows an example of how to build multi-views of the neighborhood of each item in a user behavior sequence. One challenge in searching for k-hop neighbors is that the number of such neighbors grows exponentially as k increases. This results in a more complex model as k gets larger. To address this issue, we have developed the Dirichlet weight sampling method, which allows for the selection of important k-hop neighbors while avoiding overfitting on the selected neighbors. Details will be elaborated in Section 3.2.2.

Then, for the k-th view, we use an efficient graph aggregation method to extract the local graph information of each item and apply a Transformer [192] module to capture the sequential information among all the items in the sequence $\mathcal{I}_u$ and obtain the sequence embedding from the k views. In this way, the sequence embedding can capture both the sequential behavior patterns of a specific user and the collaborative information among different users through the item dependency sub-graphs.

In the following step, we aggregate multiple views through attention network to generate a unified representation of the user behavior sequence for predicting the next potential

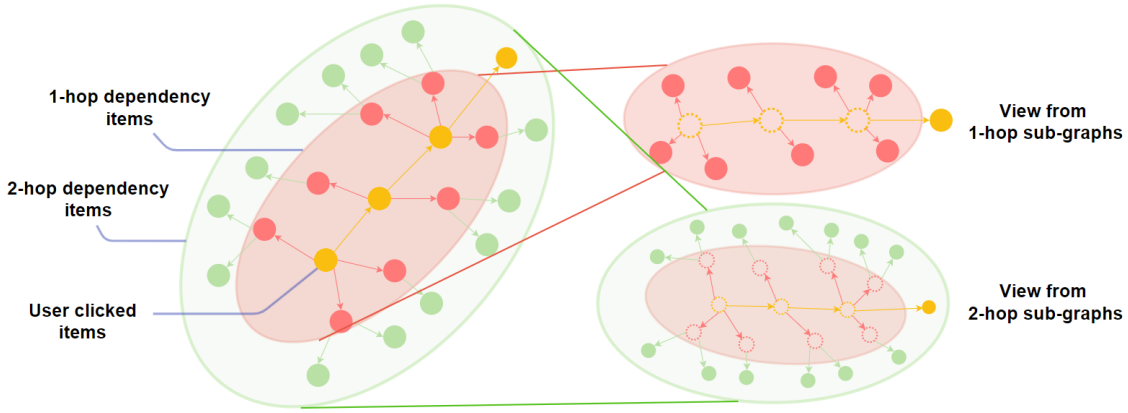


Fig. 6.3: Item dependency graph, where the yellow-colored trace represents the sequence of the user's clicked items. The red-colored nodes (1-hop neighbours) represent the items that have strong dependencies with the user-clicked items, which form the view of the first-hop neighbours. The green-colored nodes (2-hop neighbours) represent the items that have strong dependencies with the red-colored nodes, which form the view of the second-hop neighbours.

interaction items. Embeddings from different views are further aggregated using an attention network to generate a unified representation of the user behavior sequence for predicting the next potential interaction items.

6.2.3 Graph Aggregation Networks

After extracting the local sub-graph for each item in the item sequence, we aim to obtain an embedding for each item by aggregating its local dependency sub-graph information. One potential solution is to use existing GNNs, Graph Convolutional Networks (GCNs) [98] and Graph Attention Networks (GATs) [208], for message passing in the local sub-graph of each item. However, these existing GNNs are not preferred to be used in the proposed framework. Because they need to process the whole graph by adding convolutional layers or attention layers to each node of the whole graph, and obtain the next-hop embedding iteratively. Such a learning procedure is usually time-consuming (refer to Section 6.2.7 for more discussions), and could hardly process a large amount of frequently changing sub-graphs efficiently.

6.2.3.1 Intra-hop Aggregation

To improve the efficiency of message passing in the local sub-graphs of items in $\mathcal{I}_u$, we propose the following intra-hop aggregation procedure. For each item $i \in \mathcal{I}_u$, we treat it as the central node and denote its k-hop neighbors in $\mathcal{G}$ by $\mathcal{N}_i^{(k)}$. Then, we perform the k-th intra-hop aggregation via

$$\mathbf{e}_i^{(k)} = \sum_{j \in \mathcal{N}_i^{(k)}} \mathbf{T}_{i,j}^{(k)} \mathbf{e}_j, \tag{6.2}$$

where $\mathbf{e}_j$ is the initial embedding of node j. Based on Eq. (6.2), the $(k+1)$-th hop aggregation can be computed as follows,

$$\begin{aligned} \mathbf{e}_i^{(k+1)} &= \sum_{j \in \mathcal{N}_i^{(k)}} \mathbf{T}_{i,j}^{(k)} \sum_{m \in \mathcal{N}_j^{(k+1)}} \mathbf{T}_{j,m}^{(1)} \mathbf{e}_m \\ &= \sum_{j \in \mathcal{I}} \mathbf{T}_{i,j}^{(k)} \sum_{m \in \mathcal{I}} \mathbf{T}_{j,m}^{(1)} \mathbf{e}_m = \sum_{m \in \mathcal{N}_i^{(k+1)}} \mathbf{T}_{i,m}^{(k+1)} \mathbf{e}_m, \end{aligned} \tag{6.3}$$

which corresponds to Eq. (6.2) by increasing k to $k+1$. Eq. (6.3) indicates that to perform the k-th hop aggregation for $k \in \{1, 2, \cdots, K\}$, we can neglect the 1-hop to $(k-1)$-th hop sub-graphs and directly aggregate the k-th hop nodes using their dependency scores as the weights.

Compared with existing GCN-based models that concatenate k layers to process the k-th hop sub-graphs, the proposed aggregation method can directly process the specific hop of the sub-graph. Moreover, the proposed aggregation method is more suitable for the sequential recommendation task. Because the item embeddings may be frequently changed in each batch of training samples. In contrast, GCNs need to re-process the k-th hop embeddings of the entire item graph for each batch. Using GCN-based methods may significantly increase the computation time.

6.2.3.2 Dirichlet Weight Sampling

Since $\mathbf{T}_{i,j}^{(k)}$ is an empirical estimation of the overall dependency between items. For a specific user, such dependency may not reflect the exact relationships between items in the user-specific behavior sequence. Especially, when the interaction data is relatively sparse, the item dependency might be biased due to the shortage of data.

Similar issues have been reported in many related works such as SSEPT [225] and SGCN [18], where the stochastic masking is applied to the input data to avoid over-fitting due to data sparsity. Intuitively, the model should not over-fit on specific $\mathbf{T}_{i,j}^{(k)}$ on top h neighbours. To mitigate the inductive bias of the model and avoid over-fitting on transition pattern on the top h neighbours, we propose the Dirichlet sampling method to stochastically sample the transition probability, the weight parameters $\mathbf{T}_{i,j}^{(k)}$.

For a set of Dirichlet random variable $\{\mathbf{x}_1, ..., \mathbf{x}_K\} \sim \mathcal{D}(\mathbf{x}|\alpha)$, where α represents a set of hyperparameters $\{\alpha_k\}_{k=1}^K$, has the following probability density function,

$$f\left(\mathbf{x}_1, \ldots, \mathbf{x}_K; \alpha_1, \ldots, \alpha_K\right) = \frac{1}{\mathrm{B}(\alpha)} \prod_{k=1}^{K} \mathbf{x}_i^{\alpha_k - 1}, \tag{6.4}$$

where $\mathrm{B}(\cdot)$ denotes the Beta function and $\mathbf{x}_k$ denotes the Dirichlet random variable of the k-th hop neighbours. The expectation of the random variable $\{\mathbf{x}\}_{k=1}^K \sim \mathcal{D}(\mathbf{x}|\alpha)$ is $\mathbb{E}[\mathbf{x}_k] = \frac{\alpha_k}{\sum_k \alpha_k}$. Furthermore, large hyper-parameters $\{\alpha\}_{k=1}^K$ imply that $\{\mathbf{x}\}_{k=1}^K$ are concentrated around their expected value, while small $\{\alpha\}_{k=1}^K$ give high variance to $\{\mathbf{x}\}_{k=1}^K$ to make them less concentrated.

To be specific, based on the dependency score $\mathbf{T}_{i,j}^{(k)}$, we can construct the Dirichlet distribution so that the expectation of the random variable $x_{(i,j,k)}$ that follows a Dirichlet distribution equals to $\mathbf{T}_{i,j}^{(k)}$, $\mathbb{E}[x_{(i,j,k)}] = \mathbf{T}_{i,j}^{(k)}$. This Dirichlet sampling is unbiased according to $\mathbf{T}_{i,j}^{(k)}$. Finally, we use $x_{(i,j,k)}$ to replace $\mathbf{T}_{i,j}^{(k)}$ to represent the k-th hop transition probability from item i to item j in practice.

The advantage of using Dirichlet sampling to sample multinomial weight distribution rather than using stochastic masking is that Dirichlet sampling can provide randomness to the transition probability while maintaining unbiased sampling. In contrast, stochastic masking does not change the transition probability but only provides binary selection on neighbours.

6.2.4 Sequence Embedding via Transformers

For each view, we first obtain the sub-graph representation of each item in the sequence by the graph aggregation model. Then, we apply a set of transformer blocks on the item sequence to obtain the representation of the item sequence at the k-th view. For the item sequence whose length $n_u < L$, where L is the maximum length among all the sequences, we pad zero values at the beginning of the sequence to make the lengths of all the sequences to be L.

6.2.4.1 Positional Encoding

We apply learnable positional embedding $\mathbf{P} \in \mathbb{R}^{L \times d}$ referred to the previous work [92, 192], and the user's item sequence embedding at k-th view $E_{u,k}$ is performed by an

element-wise addition of item embedding and positional embedding,

$$\mathbf{E}_{u,k} = \begin{bmatrix} \mathbf{e}_{s_1}^{(k)} + \mathbf{p}_1 \\ \mathbf{e}_{s_2}^{(k)} + \mathbf{p}_2 \\ \cdots \\ \mathbf{e}_{s_L}^{(k)} + \mathbf{p}_L \end{bmatrix}. \tag{6.5}$$

6.2.4.2 Multi-head Self-attention Blocks.

We stack multiple self-attention blocks based on the embedding layer to capture the sequential feature of the user $u \in \mathcal{U}$. The conventional scaled dot-product attention for the k-th view is defined as $\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\right)\mathbf{V}$, where $\mathbf{Q} = \mathbf{E}_u\mathbf{W}^Q$ represents the query, $\mathbf{K} = \mathbf{E}_u\mathbf{W}^K$ is the key, $\mathbf{V} = \mathbf{E}_u\mathbf{W}^V$ is the value and d is the latent dimension acting as the scaled factor to regularize the dot-product value. The learnable projection matrices $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V \in \mathbb{R}^{d\times d}$ are used to project the item sequence embedding to $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$, respectively. In this work, we design the multi-view multi-head self-attention blocks, which provide multi-head attention for each view independently. Each self-attention block for the k-th view contains multiple attention heads, and the formula is defined as follows,

$$\begin{aligned} \text{MultiHeadAtt}(\mathbf{E}_{u,k}^l) &= [\text{head}_{1,k}, \text{head}_{2,k}, ..., \text{head}_{h,k}]W_k^H, \\ \text{head}_{i,k} &= \text{Attention}(\mathbf{Q}_{i,k}, \mathbf{K}_{i,k}, \mathbf{V}_{i,k}), \end{aligned} \tag{6.6}$$

where $\mathbf{Q}_{i,k} = \mathbf{E}_{u,k}^l\mathbf{W}_{i,k}^Q$, $\mathbf{K}_{i,k} = \mathbf{E}_{u,k}^l\mathbf{W}_{i,k}^K$, $\mathbf{V}_{i,k} = \mathbf{E}_{u,k}^l\mathbf{W}_{i,k}^V$ are the query, key and value. $\mathbf{W}_{i,k}^Q, \mathbf{W}_{i,k}^K, \mathbf{W}_{i,k}^V \in \mathbb{R}^{d\times d/h}$. $\mathbf{E}_{u,k}^l \in \mathbb{R}^{L\times d}$ is the input to the k-th view, l-th self-attention block. $\mathbf{W}_k^H \in \mathbb{R}^{d\times d}$ is the k-th view projection matrix to obtain the output $\mathbf{E}_{u,k}^{l+1}$, which can be the input to the next self-attention block. The output is then processed by the feed-forward layer.

Feed-Forward Layer. The feed-forward layer provides non-linearity to the transformer and supports interaction between dimensions. It consists of two affine transformations with a Gaussian Error Linear Unit (GELU) in between, to process the input data x_k from the k-th view as follows,

$$\text{FFN}(\mathbf{x}_k) = \text{GELU}(\mathbf{x}_k\mathbf{W}_{1,k} + \mathbf{b}_{1,k})\mathbf{W}_{2,k} + \mathbf{b}_{2,k}, \tag{6.7}$$

where $\mathbf{W}_{1,k}$, $\mathbf{W}_{2,k}$, $\mathbf{b}_{1,k}$, and $\mathbf{b}_{2,k}$ are learnable parameters for the k-th view which are not shared across layers.

6.2.5 Multi-view Aggregation

After performing the aforementioned steps, for each view $k \in \{0, 1, \cdots, K\}$, where $k = 0$ denotes the view from the input item sequence without item-dependency graphs, we

obtain an embedding $\mathbf{u}_k$, which is the output from the FFN layer, for a user sequence $\mathcal{I}_u$. In the following, we further merge the embeddings from all the $(K+1)$ views to generate an overall representation for each specific user to capture his/her preference and context-aware intention. The aggregation is done by using the attention mechanism via $\hat{\mathbf{u}} = \sum_{k=0}^{K} \frac{\exp(\mathbf{h}_k^\top \mathbf{u}_k)\mathbf{u}_k}{\sum_{k=0}^{K} \exp(\mathbf{h}_k^\top \mathbf{u}_k)}$, where $\hat{\mathbf{u}}$ denotes the overall embedding for user u based on the set of $(K+1)$ views, and $\mathbf{h}_k$ is a learnable parameter. It is worth noting that u_k is the output of the last FFN(x_k) layer. With the user embedding, a predicted interaction score of user u and item i is computed as, $\hat{r}_{u,i} = \hat{\mathbf{u}}\mathbf{e}_i^\top$, where $\mathbf{e}_i$ is the embedding of item i.

6.2.6 Loss Functions

We use $\mathcal{D}_s$ to represent the training set of a recommender system, where $\mathcal{D}_s := \{(r_{u,i}, u, i) | u \in \mathcal{U}, i \in \mathcal{I}\}$, and $r_{u,i}$ is the interaction score of the (u,i) pair. As our goal is to correctly predict the potential user-item interactions, the main objective of the proposed model can be formulated as follows,

$$\min_{\theta} \sum_{(u,i)\in\mathcal{D}_s} \ell_{\text{main}}(r_{u,i}, \hat{r}_{u,i}), \tag{6.8}$$

where $r_{u,i}$ is the ground-truth rating of item i given by user u, and $\hat{r}_{u,i}$ is the predicted rating. We denote $\hat{r}_{u,i} = f_\theta(\mathcal{I}_u, i)$, where f_θ is our proposed model with a set of learnable parameters θ, and $\mathcal{I}_u$ is the item sequence of user u. We adopt the binary-cross-entropy loss for ℓ_{main}, that is

$$\ell_{\text{main}} = \sum_{(u,i)\in\mathcal{D}_s} \Big[-r_{u,i} \log(\sigma(f_\theta(\mathcal{I}_u, i))) - (1 - r_{u,i}) \log(1 - \sigma(f_\theta(\mathcal{I}_u, i))) \Big]. \tag{6.9}$$

Besides the main loss in Eq. (6.9), we also aim to make use of multiple view information to construct auxiliary losses to provide more discriminative information into our model learning. As we obtain a representation of a user for each view before performing multi-view aggregation in Section 6.2.5, we can construct an individual prediction model for each view $k \in \{0, 1, \cdots, K\}$, denoted by $f_{\theta_k}(\mathcal{I}_u, i)$, where $\theta_k \subseteq \theta$ is the subset of parameters for view k. $\sigma(\cdot)$ denotes the activation function and we use the Sigmoid function similar as SASRec [92].

Thus, the final prediction of $\hat{r}_{u,i}$ can be further decomposed into a convex combination of the predicted results generated from each view: $\hat{r}_{u,i} = \sum_{k=0}^{K} w_k f_{\theta_k}(\mathcal{I}_u, i)$. Here, $w_k \in [0, 1]$ represents the attention weight of k-th view that can be obtained by the attention mechanism as elaborated in Section 3.5, and $f_{\theta_k}(\mathcal{I}_u, i)$ denotes prediction score with the k-th view's transformer model. Therefore, the binary-cross-entropy loss can be re-written as follows,

$$\ell_{\text{main}} = \sum_{(u,i)\in\mathcal{D}_s} \Big[-r_{u,i} \log\Big(\sigma\Big(\sum_k w_k f_{\theta_k}(\mathcal{I}_u, i)\Big)\Big) - (1 - r_{u,i}) \log\Big(1 - \sigma\Big(\sum_k w_k f_{\theta_k}(\mathcal{I}_u, i)\Big)\Big)\Big]. \tag{6.10}$$

However, the above loss function causes biased updates for each individual model: the lower the attention weight, the lesser the individual view's model f_{θ_k} is optimized. To solve this issue, we propose an auxiliary loss ℓ_{indiv} that focuses on updating the network parameters in each individual view, shown as follows,

$$\ell_{\text{indiv}} = \sum_{k} \sum_{(u,i)\in\mathcal{D}_s} \left[-r_{u,i}\log(\sigma(f_{\theta_k}(\mathcal{I}_u, i))) - (1 - r_{u,i})\log(1 - \sigma(f_{\theta_k}(\mathcal{I}_u, i)))\right]. \tag{6.11}$$

6.2.6.1 Mutual Information Maximization

Note that the auxiliary loss function in Eq. (6.11) can be optimized by minimizing each individual loss independently. However, minimizing Eq. (6.11) or Eq. (6.9) do not guarantee the consensus of each view. Thus, we encode the consensus constraint into the predictions of different individual views on the same user-item pair, by introducing another objective to maximize the mutual information between different views. We propose to maximize the mutual information among the multiple views as follows,

$$\max_{\theta} \mathbb{E}_{\mathbf{u}_k} I_\theta(\mathbf{u}_k; \mathbf{U} - \mathbf{u}_k), \tag{6.12}$$

where $\mathbf{u}_k$ is the representations from the k-th view, $k \in \{0, ..., K\}$.
$\mathbf{U} - \mathbf{u}_k = \{\mathbf{u}_0, ..., \mathbf{u}_{k-1}, \mathbf{u}_{k+1}, ..., \mathbf{u}_K\}$ denotes the representations from all views except $\mathbf{u}_k$, and $I_\theta(\mathbf{u}_k; \mathbf{U} - \mathbf{u}_k) = \sum_{\mathbf{u}_k} p_\theta(\mathbf{u}_0, ..., \mathbf{u}_K)\log\frac{p_\theta(\mathbf{u}_k|\mathbf{U}-\mathbf{u}_k)}{p_\theta(\mathbf{u}_k)}$ computes the mutual information between the view $\mathbf{u}_k$ and other views. By maximizing Eq. (6.12), we expect that the multiple views will converge to a consistent representation of the user's preference. However, the mutual information is difficult to compute. As $I_\theta(\mathbf{u}_k, \mathbf{U} - \mathbf{u}_k) \propto \frac{p(\mathbf{u}_k|\mathbf{U}-\mathbf{u}_k)}{p(\mathbf{u}_k)}$, following [156], we minimize the noise contrastive loss as follows,

$$\ell_{\text{contrast}} = -\frac{1}{K+1}\sum_{k=0}^{K} \mathbb{E}_{\mathbf{u}_k}\log\frac{f(\mathbf{u}_k, \mathbf{U} - \mathbf{u}_k)}{f(\mathbf{u}_k, \mathbf{U} - \mathbf{u}_k) + \sum_{\mathbf{u}_{neg}} f(\mathbf{u}_{neg}, \mathbf{U} - \mathbf{u}_k)}, \tag{6.13}$$

where $\mathbf{u}_{neg}$ is the sampled negative view's representations from the batch. $f(\mathbf{u}_k, \mathbf{U}-\mathbf{u}_k) = \text{Sigmoid}(\langle \mathbf{u}_k, \frac{\sum_{j\neq k}\mathbf{u}_j}{K}\rangle)$ denotes the function used to calculate the similarity of the view $\mathbf{u}_k$ to other views, and $\langle\cdot\,,\cdot\rangle$ denotes the inner product. In our methodology, negative views are derived from the behavior sequences of users within the same batch, excluding the target user. For each user, the total number of negative samples is $(K+1)\times(b-1)$, where b is the batch size in training. Since the number of total negative samples is relatively large, in practice, for each user in each view, we randomly sample b negative samples from the negative sample set with size $(K+1)\times(b-1)$.

Proposition 1. *Suppose $\ell_{contrast}$ is driven in Eq.* (6.13) *and let N be the total number of training samples from the training set, the following inequality holds,*

$$\ell_{contrast} \geq -\mathbb{E}_{\mathbf{u}_k} I_\theta(\mathbf{u}_k; \mathbf{U} - \mathbf{u}_k) + \log(N). \tag{6.14}$$

Proof. Since the optimal value of $I(\mathbf{u}_k, \mathbf{U} - \mathbf{u}_k)$ is given by $\frac{p(\mathbf{u}_k|\mathbf{U}-\mathbf{u}_k)}{p(\mathbf{u}_k)}$, due to the proportional property as we discussed in Section 3.6.1, we can insert the ratio back to Eq. (6.13) and obtain:

$$\begin{aligned}
\ell_{\text{contrast}} &= -\frac{1}{K+1}\sum_{k=0}^{K}\mathbb{E}_{\mathbf{u}_k}\log\left[\frac{\frac{p(\mathbf{u}_k|\mathbf{U}-\mathbf{u}_k)}{p(\mathbf{u}_k)}}{\frac{p(\mathbf{u}_k|\mathbf{U}-\mathbf{u}_k)}{p(\mathbf{u}_k)}+\sum_{\mathbf{u}_{\text{neg}}}\frac{p(\mathbf{u}_{\text{neg}}|\mathbf{U}-\mathbf{u}_k)}{p(\mathbf{u}_{\text{neg}})}}\right] \\
&= \frac{1}{K+1}\sum_{k=0}^{K}\mathbb{E}_{\mathbf{u}_k}\log\left[1+\frac{p\left(\mathbf{u}_k\right)}{p\left(\mathbf{u}_k \mid \mathbf{U}-\mathbf{u}_k\right)}\sum_{\mathbf{u}_{\text{neg}}}\frac{p\left(\mathbf{u}_{\text{neg}} \mid \mathbf{U}-\mathbf{u}_k\right)}{p\left(\mathbf{u}_{\text{neg}}\right)}\right] \\
&\approx \frac{1}{K+1}\sum_{k=0}^{K}\mathbb{E}_{\mathbf{u}_k}\log\left[1+\frac{p\left(\mathbf{u}_k\right)}{p\left(\mathbf{u}_k \mid \mathbf{U}-\mathbf{u}_k\right)}(N-1)\mathop{\mathbb{E}}_{\mathbf{u}_{\text{neg}}}\frac{p\left(\mathbf{u}_{\text{neg}} \mid \mathbf{U}-\mathbf{u}_k\right)}{p\left(\mathbf{u}_{\text{neg}}\right)}\right] \\
&= \frac{1}{K+1}\sum_{k=0}^{K}\mathbb{E}_{\mathbf{u}_k}\log\left[1+\frac{p\left(\mathbf{u}_k\right)}{p\left(\mathbf{u}_k \mid \mathbf{U}-\mathbf{u}_k\right)}(N-1)\right] \\
&\geq \frac{1}{K+1}\sum_{k=0}^{K}\mathbb{E}_{\mathbf{u}_k}\log\left[\frac{p\left(\mathbf{u}_k\right)}{p\left(\mathbf{u}_k \mid \mathbf{U}-\mathbf{u}_k\right)}N\right] \\
&= -\mathbb{E}_{\mathbf{u}_k} I\left(\mathbf{u}_i; \mathbf{U}-\mathbf{u}_k\right) + \log(N).
\end{aligned} \tag{6.15}$$

This proof partially refers to the ideas from the literature [156]. It shows that minimizing Eq. (6.13) results in maximizing the lower bound of the mutual information of each view $\mathbf{u}_k$ to the other views $\{\mathbf{U}-\mathbf{u}_k\}$, $k \in \{0, ..., K\}$. Note that a pairwise multi-view contrastive learning model was proposed in [203]. However, the time complexity of the pairwise contrastive model grows exponentially with the number of views, which is not practical when the number of views is large. In contrast, the complexity of Eq. (6.13) is linear with the number of views.

Overall loss. To sum up, we combine the main loss function ℓ_{main} in Eq. (6.9), the sum of individual losses from different views ℓ_{indiv} in Eq. (6.11), and the contrastive loss ℓ_{contrast} in Eq. (6.13) to construct an overall loss to train the proposed model,

$$\ell_{\text{combine}} = \ell_{\text{main}} + \lambda_1 \ell_{\text{indiv}} + \lambda_2 \ell_{\text{contrast}}, \tag{6.16}$$

where λ_1 and λ_2 are the hyper-parameters used to control the weights of the regularization components.

6.2.7 Time Complexity Discussion

We compare the training time complexity of the proposed model with existing GNN-based models for sequential recommendation. To compare the complexity under the sequential recommendation setting, we apply GNN models to process the user-item graph, obtain the embeddings of items, and input the embeddings into the transformer model. Suppose we have a graph $\mathcal{G}$ with n nodes and e edges in total. We train the model with batch size s. The number of training samples is M. The number of neighbours being sampled for each node is h. The number of GNN layers is K. The time complexity for a fixed-length transformer is constant T. The dimension of embedding size is d. Thus, the time complexity of the proposed model in each epoch is $O(hKdM + MKT)$, where hK represents the intra-hop aggregation. The proposed model's time complexity linearly increases with the expansion of the number of sampled neighbours h, and the sub-graph size K.

To show the time complexity comparison of our model with the state-of-the-art graph-based models, such as GCN [98], GraphSage [67], FastGCN [19], and Cluster-GCN [21], in the same sequential recommendation settings, we take the aforementioned models as the graph-backbone. For each user behavior, the graph-backbone takes the user-item interaction graph as the input and outputs the embedding of that behavior. Then, we input the sequence of behavior embeddings into a transformer model to predict the next behavior of that user. In Table 6.2, we show the comparison of the training complexity of our model with the aforementioned models. Moreover, $K \ll h \ll n \ll m$, and $b \ll n$ are in common recommendation scenarios. Therefore, compared to the GCN+SR (SR stands for sequential recommendation with transformers), GraphSage+SR, FastGCN+SR and Cluster-GCN+SR, our model performs significantly faster. Furthermore, we record the time spent of the aforementioned methods for training one batch of data from the ML-1M dataset in Table 6.2, with the batch size of 256 and the hidden dimension of 128 on a single 1080Ti GPU. It is worth noting that our model achieves 6.2x and 8.1x faster compared to FastGCN + transformers and GCN + transformers respectively.

The reduction in time complexity is primarily attributed to the use of our proposed multi-hop graph aggregation networks. The graph aggregation is primarily achieved through multi-hop aggregation, wherein the main computation involves finding k-hop neighbors, which can be performed during the pre-processing phase. Additionally, employing sparse multi-hop aggregation can boost performance speed for rapidly changing graphs, where the embedding of each graph node is updated after every training step.

Moreover, the utilization of the Dirichlet sampling method does not increase the complexity of the training process, as it primarily influences the sampling weights of each k-hop neighbor and does not add computation cost to the model. In comparison to other baseline methods, our multi-hop graph aggregation networks can be easily integrated into other backbone models, which we will discuss in detail in Section 5.1.5.

Overall, our proposed model boasts numerous advantages, such as reduced time complexity, increased performance speed, adaptability to various backbone models, and the potential for enhanced recommendation accuracy. The incorporation of the Dirichlet

Table 6.2: Time complexity comparison of training each epoch for GNN+SR (sequential recommendation) training algorithms. n is the total number of nodes. M is the total number of training samples. m is the total number of edges. K is the number of layers. b is the batch size. h is the number of neighbors being sampled for each node. For simplicity, the dimensions of the node hidden features remain constant, denoted by d. The time complexity for processing each fixed length sequence by the transformer is constant, denoted by T.

Model	Time Complexity	Time/batch
GCN [98] with sequential recommendation	$O\left(\frac{(Kmd+Knd^2)M}{b}+MT\right)$	0.5347s
GraphSage [67] with sequential recommendation	$O\left(\frac{(h^K nd^2)M}{b}+MT\right)$	23.4390s
FastGCN [19] with sequential recommendation	$O\left(\frac{(Khnd^2)M}{b}+MT\right)$	0.4126s
Cluster-GCN [21] with sequential recommendation	$O\left(\frac{(Kmd+Knd^2)M}{b}+MT\right)$	0.5119s
Ours	$O(hKdM+MKT)$	0.0659s

sampling method enables our approach to strike a balance between exploration and exploitation during the training process, resulting in more stable robustness. The combination of these benefits, along with the potential for wider applications across different domains, establishes our proposed method as a promising solution for recommendation systems. In the following section, we will elaborate on extensive experiments to further evaluate our model's performance and showcase the effectiveness of our proposed approach.

6.3 Experiments

In this section, we present comprehensive experiments to showcase the advantages and effectiveness of our model and its individual components. We assess our model using five publicly available datasets: MovieLens-1M [1], Yelp [2], Amazon Video Games [3], Amazon CDs [4], and Tmall [5]. MovieLens datasets serve as widely recognized benchmarks for recommendation systems, containing movie reviews from a prominent movie review website. The Amazon Video Games and Amazon CDs datasets comprise user reviews for products in the video game and CD categories on Amazon, respectively. Meanwhile, the Yelp dataset features user ratings for various businesses on Yelp. Lastly, the Tmall dataset includes user feedback from the e-commerce platforms Tmall and Taobao. Table 6.3 provides detailed statistics for each dataset.

[1] https://grouplens.org/datasets/movielens/1m/
[2] https://www.yelp.com/dataset
[3] https://jmcauley.ucsd.edu/data/amazon/
[4] https://jmcauley.ucsd.edu/data/amazon/
[5] https://tianchi.aliyun.com/dataset/dataDetail?dataId=53

Table 6.3: Statistics of the experimented data.

Dataset	User #	Item #	Interaction #	Avg. actions/user #
Yelp	25,677	25,815	0.7M	26.5
Movielens-1M	6,040	3,707	1.0M	163.5
Video Game	24,303	10,674	207k	8.51
CD	75,258	64,444	1.0M	13.3
Tmall	48,618	40,729	335k	6.9

6.3.1 Evaluation Metrics

To evaluate the performance of the recommendation models, we adopt the commonly used leave-one-out evaluation, i.e. next-item prediction task [77, 92]. We split the dataset into train, validation and test set: for each user, we take the last item of the behavior sequence as the test set, the second-last as the validation set, and the rest as the train set. We evaluate all the methods in terms of *Hit Ratio* (HR) and *Normalized Discounted Cumulative Gain* (NDCG). HR@N measures the average number of positive items being retrieved in the generated top-N recommendation list for each user. NDCG@N extends HR@N by considering the positions of retrieved positive items in the top-N recommendation list. A higher value in the metric reflects a higher performance.

In this work, we opt for the full-ranking strategy over the sampling-based ranking strategy when evaluating recommendation performance, as suggested by [105]. Specifically, for each user, we sort all the candidate items in descending order based on the predicted scores and choose N top-ranked items as the top-N recommendation list. In the experiments, we empirically set N to 5, 10 and 20. As we only have one ground truth item for each user in the testing data, HR@N is equivalent to Recall@N and proportional to Precision@N.

6.3.2 Baseline Methods

To show the effectiveness of our method, we include two groups of recommendation baselines [6]: graph-based recommendation methods, including:

- (a) Light-GCN [74] uses Graph Convolutional Networks to learn users' preferences on items from the bipartite user-item interaction graph.

- (h) DHCN [229] designs Hypergraph Convolutional Networks to capture both session graph and global graph information.

- (i) SUGER [16] proposes Interest-fusion Graph Convolutional Layers and Interest-extraction Graph Pooling Layers to extract long/short-term interest for sequential prediction.

[6]The serial numbers of the methods correspond to Table 6.4.

- (j) GES [262] conducts graph convolution on the hybrid item graph to generate smoothed item embeddings.
- (l) SRJGraph [252] proposes a GNN-based CTR model to apply to both search and recommendation scenarios.
- (m) SAPL [159] designs a reinforcement learning strategy for learning users' sentiments on items to correct recommendations.
- (n) ReMR [221] adopts a reinforcement learning framework for multi-level recommendation reasoning.
- (r) KERL [217] fuses knowledge graph information into a RL framework for sequential recommendation.

We also adopt the following well-known sequential recommendation methods which are not based on graph architectures:

- (b) Caser [198] employs CNN in both horizontal and vertical ways to model high-order Markov chains of users' behaviors.
- (c) SASRec [92] employs a transformer model to capture the user's sequential behavior and predict the next item for the recommendation.
- (d) GRU4REC [78] adopts Recurrent Neural Networks with multiple GRU layers to sequentially predict users' next behaviors.
- (e) HGN [140] hierarchically combines gating networks to capture user's intentions based on the user's features and sequential behaviors.
- (f) Bert4Rec [192] is based on SARec [92] architecture. It uses the BERT [34] architecture instead of transformers to extract user's intentions and conduct sequential predictions.
- (g) STOSA [41] uses the Wasserstein self-attention networks to capture users' behavior patterns from their sequential behavior histories. It is one of the state-of-the-art non-graph-based sequential recommendation methods.
- (k) DIEN [256] designs the interest evolution network for CTR prediction.
- (o) RKSA [87] is a sequential recommendation model which uses relation-aware kernelized self-attention to enhance the prediction.
- (p) MT4SR [41] adopts a multi-relational transformer to capture the auxiliary item relationships in sequential recommendations.
- (q) DFAR [123] uses a dual-interest disentangling method to factorize the relation between different types of feedback and decouple positive and negative interests before performing disentanglement on their representations.

Table 6.4: Recommendation performance achieved by different methods in terms of HR and NDCG on the Yelp dataset. The best results are in **boldface**, and the second best results are underlined. The improvements achieved by our model over baseline methods are significant with p-value smaller than 0.001.

Methods	Yelp					
	HR@5	NDCG@5	HR@10	NDCG@10	HR@20	NDCG@20
(a) Light-GCN	0.0191	0.0118	0.0395	0.0174	0.0586	0.0225
(b) Caser	0.0185	0.0110	0.0361	0.0178	0.0597	0.0239
(c) SASRec	0.0224	0.0138	0.0425	0.0214	0.0689	0.0281
(d) GRU	0.0196	0.0118	0.0343	0.0165	0.0558	0.0220
(e) HGN	0.0231	0.0147	0.0401	0.0198	0.0652	0.0261
(f) BERT4Rec	0.0198	0.0113	0.0387	0.0189	0.0599	0.0241
(g) STOSA	0.0223	0.0138	0.0414	0.0204	0.0657	0.0265
(h) DHCN	0.0179	0.0112	0.0319	0.0155	0.0532	0.0208
(i) SURGE	0.0239	0.0113	0.0429	0.0219	0.0692	0.0288
(j) GES	0.0187	0.1106	0.0343	0.0203	0.0596	0.0246
(k) DIEN	0.0205	0.0107	0.0371	0.0187	0.0674	0.0268
(l) SRJGraph	0.0204	0.0131	0.0421	0.0208	0.0675	0.0277
(m) SAPL	0.0257	0.0161	<u>0.0446</u>	0.0217	0.0716	<u>0.0287</u>
(n) ReMR	0.0248	0.0156	0.0437	0.0215	0.0711	0.0283
(o) RKSA	0.0231	0.0147	0.0438	0.0218	0.0694	0.0282
(p) MT4SR	<u>0.0258</u>	<u>0.0165</u>	0.0440	<u>0.0221</u>	<u>0.0723</u>	0.0284
(q) DFAR	0.0229	0.0145	0.0429	0.0213	0.0691	0.0276
(r) KERL	0.0224	0.0138	0.0424	0.0210	0.0692	0.0279
(s) Ours	**0.0276**	**0.0174**	**0.0453**	**0.0230**	**0.0738**	**0.0299**

6.3.3 Experiment Settings

For a fair comparison, we adopt the code provided by the corresponding authors or implement the method if the code is not provided. All other hyper-parameters and initialization strategies are according to the suggestions of the methods' authors. We also tune the parameters to make the baseline methods perform well on different datasets using the validation set. We implement our model using PyTorch and train the model using Adam optimizer with a learning rate of 0.001. We set the maximum sequence length $L = 100$ for all datasets. We choose graph size from $\{1, 2, 3\}$, the number of transformer layers from $\{1, 2, 3\}$ and the number of attention heads from $\{1, 2, 4\}$. We vary the embedding size in $\{32, 64, 128\}$. We further conduct paired per-user significance tests according to the method in the literature [179], verifying that all the improvements are statistically significant for $p < 0.001$. All models are trained on the NVIDIA GeForce GTX 1080 Ti GPU.

Table 6.5: Recommendation performance achieved by different methods in terms of HR and NDCG on the ML-1M dataset. The best results are in **boldface**, and the second best results are underlined. The improvements achieved by our model over baseline methods are significant with p-value smaller than 0.001.

Methods	ML-1M					
	HR@5	NDCG@5	HR@10	NDCG@10	HR@20	NDCG@20
(a) Light-GCN	0.1145	0.0695	0.1610	0.0857	0.2631	0.1146
(b) Caser	0.1272	0.0808	0.2017	0.1048	0.2930	0.1261
(c) SASRec	0.1288	0.0805	0.1877	0.0950	0.2611	0.1153
(d) GRU	0.0984	0.0635	0.1522	0.0804	0.2341	0.1012
(e) HGN	0.1281	0.0825	0.1899	0.0964	0.2836	0.1231
(f) BERT4Rec	0.1152	0.0712	0.1658	0.0894	0.2345	0.1091
(g) STOSA	0.1302	0.0826	0.1881	0.1013	0.2741	0.1221
(h) DHCN	0.1290	0.0812	0.1697	0.0941	0.2329	0.1025
(i) SURGE	0.1378	0.0875	0.2037	0.1102	0.2957	0.1278
(j) GES	0.1320	0.0815	0.1737	0.0955	0.2474	0.1085
(k) DIEN	0.1175	0.0754	0.1792	0.0912	0.2540	0.1106
(l) SRJGraph	0.1217	0.0784	0.1817	0.0927	0.2618	0.1147
(m) SAPL	0.1483	<u>0.0942</u>	0.2247	0.1086	0.3017	0.1338
(n) ReMR	0.1451	0.0931	0.2135	0.1050	0.2986	0.1314
(o) RKSA	0.1357	0.0824	0.1978	0.1024	0.2847	0.1201
(p) MT4SR	<u>0.1507</u>	0.0869	<u>0.2314</u>	<u>0.1132</u>	<u>0.3124</u>	<u>0.1411</u>
(q) DFAR	0.1368	0.0841	0.1982	0.1031	0.2862	0.1224
(r) KERL	0.1371	0.0836	0.1994	0.1047	0.2934	0.1253
(s) Ours	**0.1629**	**0.1075**	**0.2519**	**0.1352**	**0.3620**	**0.1635**

Table 6.6: Recommendation performance achieved by different methods in terms of HR and NDCG on the Video Games dataset. The best results are in **boldface**, and the second best results are underlined. The improvements achieved by our model over baseline methods are significant with p-value smaller than 0.001.

Methods	Video Games					
	HR@5	NDCG@5	HR@10	NDCG@10	HR@20	NDCG@20
(a) Light-GCN	0.0171	0.0116	0.0265	0.0152	0.0459	0.0201
(b) Caser	0.0182	0.0126	0.0281	0.0156	0.0455	0.0198
(c) SASRec	0.0573	0.0352	0.0836	0.0435	0.1238	0.0554
(d) GRU	0.0436	0.0277	0.0705	0.0361	0.1109	0.0463
(e) HGN	0.0456	0.0293	0.0760	0.0391	0.1177	0.0492
(f) BERT4Rec	0.0555	0.0348	0.0815	0.0425	0.1185	0.0535
(g) STOSA	0.0602	0.0391	0.0962	0.0501	<u>0.1427</u>	<u>0.0622</u>
(h) DHCN	0.0470	0.0295	0.0794	0.0399	0.1222	0.0507
(i) SURGE	0.0587	0.0378	0.0875	0.0415	0.1246	0.0542
(j) GES	<u>0.0618</u>	0.0375	<u>0.1031</u>	<u>0.0503</u>	0.1421	0.0614
(k) DIEN	0.0528	0.0334	0.0803	0.0412	0.1185	0.0506
(l) SRJGraph	0.0514	0.0327	0.0756	0.0392	0.1071	0.0459
(m) SAPL	0.0608	0.0381	0.0912	0.0485	0.1419	0.0610
(n) ReMR	0.0601	0.0374	0.0903	0.0480	0.1397	0.0601
(o) RKSA	0.0586	0.0367	0.0849	0.0455	0.1281	0.0572
(p) MT4SR	0.0614	<u>0.0389</u>	0.0916	0.0487	0.1421	0.0608
(q) DFAR	0.0593	0.0387	0.0862	0.0476	0.1304	0.0595
(r) KERL	0.0598	0.0381	0.0877	0.0496	0.1327	0.0598
(s) Ours	**0.0650**	**0.0416**	**0.1035**	**0.0536**	**0.1577**	**0.0669**

Table 6.7: Recommendation performance achieved by different methods in terms of HR and NDCG on the CD dataset. The best results are in **boldface**, and the second best results are underlined. The improvements achieved by our model over baseline methods are significant with p-value smaller than 0.001.

Methods	CD					
	HR@5	NDCG@5	HR@10	NDCG@10	HR@20	NDCG@20
(a) Light-GCN	0.0098	0.0061	0.0152	0.0079	0.0254	0.0105
(b) Caser	0.0086	0.0053	0.0145	0.0072	0.0248	0.0098
(c) SASRec	0.0342	0.0203	0.0594	0.0272	0.0814	0.0364
(d) GRU	0.0084	0.0053	0.0144	0.0072	0.0239	0.0096
(e) HGN	0.0289	0.0187	0.0456	0.0239	0.0697	0.0303
(f) BERT4Rec	0.0335	0.0195	0.0568	0.0252	0.0795	0.0332
(g) STOSA	0.0324	0.0215	0.0494	0.0271	0.0742	0.0331
(h) DHCN	0.0176	0.0107	0.0298	0.0147	0.0476	0.0191
(i) SURGE	0.0315	0.0205	0.0482	0.0261	0.0722	0.0337
(j) GES	0.0351	0.0202	0.0620	0.0282	0.0971	0.0371
(k) DIEN	0.0304	0.0181	0.0567	0.0245	0.0779	0.0338
(l) SRJGraph	0.0317	0.0194	0.0574	0.0263	0.0792	0.0347
(m) SAPL	0.0407	0.0253	0.0649	0.0324	0.0925	0.0412
(n) ReMR	0.0390	0.0243	0.0638	0.0316	0.0901	0.0402
(o) RKSA	0.0367	0.0224	0.0613	0.0289	0.0847	0.0384
(p) MT4SR	0.0391	0.0246	0.0643	0.0317	0.0918	0.0412
(q) DFAR	0.0382	0.0237	0.0631	0.0296	0.0869	0.0390
(r) KERL	0.0385	0.0241	0.0621	0.0295	0.0858	0.0392
(s) Ours	**0.0439**	**0.0283**	**0.0682**	**0.0360**	**0.1004**	**0.0440**

Table 6.8: Recommendation performance achieved by different methods in terms of HR and NDCG on the Tmall dataset. The best results are in **boldface**, and the second best results are underlined. The improvements achieved by our model over baseline methods are significant with p-value smaller than 0.001.

Methods	Tmall					
	HR@5	NDCG@5	HR@10	NDCG@10	HR@20	NDCG@20
(a) Light-GCN	0.1741	0.1432	0.2047	0.1605	0.2323	0.1792
(b) Caser	0.1399	0.1233	0.1544	0.1280	0.1678	0.1314
(c) SASRec	0.3549	0.2749	0.4347	0.3047	0.4747	0.3149
(d) GRU	0.2741	0.2394	0.3004	0.2480	0.3253	0.2543
(e) HGN	0.1324	0.1071	0.1610	0.1156	0.1910	0.1232
(f) BERT4Rec	0.3349	0.2618	0.4275	0.2975	0.4579	0.3067
(g) STOSA	0.4039	0.3011	0.4334	0.3308	0.4802	0.3326
(h) DHCN	0.4126	0.3175	0.4576	0.3329	0.5164	0.3467
(i) SURGE	0.3743	0.2847	0.4480	0.3146	0.4848	0.3185
(j) GES	0.4014	0.3090	0.4565	0.3345	0.5108	0.3380
(k) DIEN	0.3067	0.2473	0.3961	0.2758	0.4536	0.2904
(l) SRJGraph	0.2935	0.2278	0.3847	0.2646	0.4419	0.2851
(m) SAPL	0.3930	0.3023	0.4584	0.3206	0.5036	0.3317
(n) ReMR	0.3859	0.2981	0.4512	0.3168	0.4954	0.3253
(o) RKSA	0.3692	0.2847	0.4453	0.3124	0.4821	0.3196
(p) MT4SR	0.3970	0.3012	0.4602	0.3225	0.5127	0.3380
(q) DFAR	0.3758	0.2931	0.4503	0.3152	0.4921	0.3247
(r) KERL	0.3727	0.2914	0.4531	0.3209	0.4952	0.3277
(s) Ours	**0.4169**	**0.3236**	**0.4785**	**0.3433**	**0.5313**	**0.3583**

6.4 Results and Analysis

The experiment results for all the methods on the five datasets with HR@N and NDCG@N are shown in Table 6.4, Table 6.5, Table 6.6, Table 6.7, and Table 6.8 respectively. From the results, we can infer that, in general, our sequential recommendation method can outperform the state-of-the-art sequential recommendation methods. The results show that our method outperforms all the baselines in various datasets. The performance of our method largely improves on datasets ML-1M and Yelp. On Yelp, our model achieved 0.0738 on HR@20, compared to SOTA graph-based sequential recommendation models such as SURGE which is 0.0692 on HR@20 and GES which is 0.0596 on HR@20, our model achieves 6.7% and 23.8% improvements. Since ML-1M and Yelp datasets are based on user's reviews, the connections between items are more correlated and therefore, a large amount of information can be stored in the item-to-item connections. Therefore, applying items' sub-graphs can help improve the model's representation capabilities. For e-commerce datasets, e.g. Amazon CD and Video Games, we can observe relatively high improvements in the HR and NDCG scores. In the Tmall dataset, the improvement is relatively limited, since in this dataset we consider each session as an independent user and the user histories are sparse, the user preference is relatively random with respect to the item dependency due to the sparsity.

6.4.1 Ablation Study

We perform ablation studies on our model by showing how different components of our model can affect the recommendation performance in the following aspect:

- The usage of item dependency sub-graphs: whether the sub-graph improves the recommendation accuracy and how the sub-graph size affects the model performance.
- Dirichlet sampling: how the Dirichlet sampling with hyper-parameter α affects the performance.
- Hyperparameters: how the hyper-parameters, the embedding size, λ_1 and λ_2, affect the model's performance.

We also investigate two advantage aspects of our model:

- The adaptation of our proposed graph-based multi-view model: can our model be adapted to other sequential user behavior encoders except for the transformer networks?
- Can other types of item correlations be adopted in our model?

Table 6.9: Ablation study for HR@10 and NDCG@10 with different sub-graph size. 0-hop refers to the SASRec method.

Dataset	Yelp		ML-1M		Video Games		CD	
Metric	HR@10	NDCG@10	HR@10	NDCG@10	HR@10	NDCG@10	HR@10	NDCG@10
0-hop	0.0425	0.0214	0.1877	0.0950	0.0836	0.0435	0.0594	0.0272
1-hop	**0.0454**	0.0225	0.2434	0.1330	0.0942	0.0478	0.0655	0.0341
2-hop	0.0453	**0.0230**	**0.2583**	**0.1409**	**0.1068**	**0.0543**	**0.0682**	**0.0360**
3-hop	0.0450	0.0218	0.2519	0.1297	0.1010	0.0527	0.0671	0.0350

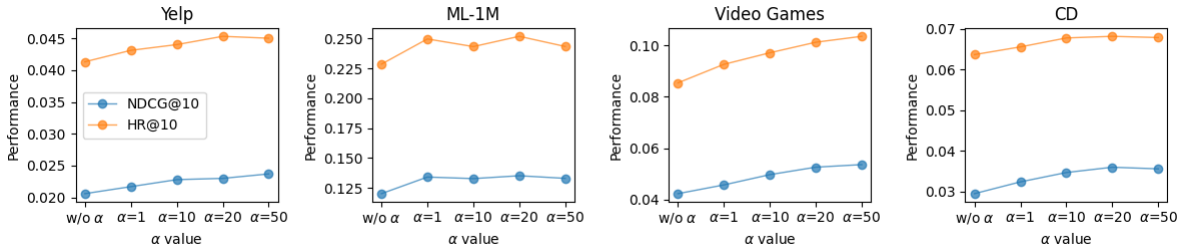


Fig. 6.4: Ablation study for HR@10 and NDCG@10 with different α values

6.4.1.1 Impact of Using Item Dependency Sub-graphs

To justify the effectiveness of applying item sub-graphs to the sequential recommendation, we show how the sub-graph's size can affect the performance of our model. The results of using different sub-graph sizes on the four datasets are shown in Table 6.9. From the experiment, we can infer that with the sub-graphs, our model outperforms the SASRec model, which does not employ an item dependency graph to help the model make predictions. In addition, our model achieves high performance at around 2-hop sub-graph size. Using too large sub-graphs may hinder the performance, since when the sub-graph is too large, the dependencies between the centre item and border items of the sub-graph are not informative enough for predicting users' preferences.

6.4.1.2 Impact Dirichlet Sampling Parameter α

The Dirichlet Sampling Parameter $\{\alpha\}_{k=1}^{K}$ controls the variance of the randomly sampled dependency scores $\mathbf{T}_{i,j}^{(k)}$, which serves as an important hyper-parameter to mitigate the inductive bias and enhance our model's prediction accuracy.

Figure 6.4 shows the performance of our 2-hop sub-graph model with different Dirich-

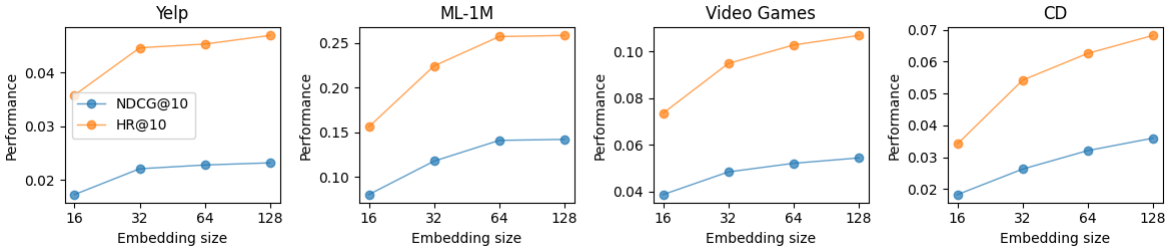


Fig. 6.5: Ablation study for HR@10 and NDCG@10 with different embedding size: 16, 32, 64, and 128.

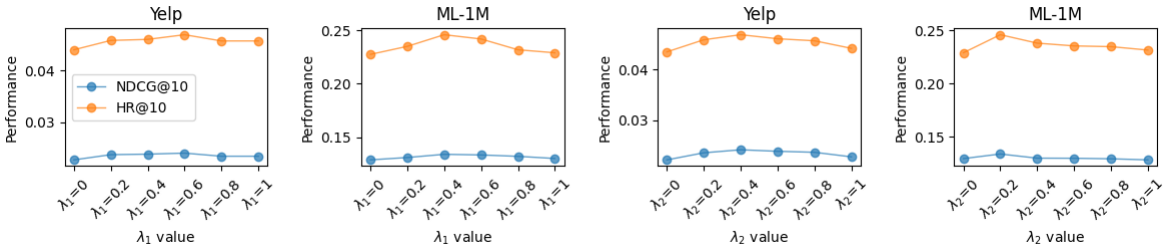


Fig. 6.6: Ablation study for HR@10 and NDCG@10 with different λ_1 and λ_2 values. The left two figures shows the performance for different λ_1 values, and the right two figures shows the performance for different λ_2 values.

Table 6.10: Ablation study on the adaptability of our model. We fit our proposed multi-view graph model into different encoder networks: HGN, GRU, and Caser.

Dataset	Yelp		ML-1M		Video Games		CD	
Metric	HR@10	NDCG@10	HR@10	NDCG@10	HR@10	NDCG@10	HR@10	NDCG@10
Transformer	0.0425	0.0214	0.1877	0.0950	0.0836	0.0435	0.0594	0.0272
Transformer + graph multi-view	**0.0454**	**0.0230**	**0.2583**	**0.1409**	**0.1068**	**0.0543**	**0.0682**	**0.0360**
HGN	0.0401	0.0198	0.1899	0.0964	0.0760	0.0391	0.0456	0.0239
HGN + graph multi-view	**0.0428**	**0.0212**	**0.2081**	**0.1083**	**0.0816**	**0.0415**	**0.0483**	**0.0246**
GRU	0.0343	0.0165	0.1522	0.0804	0.0705	0.0361	0.0144	0.0072
GRU + graph multi-view	**0.0368**	**0.0179**	**0.1710**	**0.0927**	**0.0758**	**0.0380**	**0.0249**	**0.0127**
Caser	0.0361	0.0178	0.2018	0.1048	0.0280	0.0156	0.0145	0.0072
Caser + graph multi-view	**0.0382**	**0.0185**	**0.2246**	**0.1137**	**0.0492**	**0.0251**	**0.0286**	**0.0146**

let sampling parameters α. We calculate the Dirichlet parameter for each item in the sub-graph as $\alpha_j^{(k)} = \alpha \cdot \mathbf{T}_{i,j}^{(k)}$ for items j in the k-th hop of the sub-graph centred around item i. During the training, the items transition probability $\mathbf{T}$ is sampled from $\mathcal{D}(\alpha_1, ..., \alpha_K)$. With high α, the sampling is more concentrated around the expectation of the Dirichlet distribution, which is the true weight distribution obtained from the item-item dependency graph. From the results, we can infer that by creating perturbations to the item-item dependency graph, the model can achieve higher accuracy in the test set than the model without perturbations. Meanwhile, the perturbations need to be concentrated around the empirical estimation of the transition probability $\mathbf{T}_{i,j}^{(k)}$ to reduce the inaccuracy of the sampling.

6.4.1.3 Impact of the dimensionality of hidden variables

We conduct experiments with embedding dimensionality from $\{16, 32, 64, 128\}$, to test the robustness and better understand the effect of dimensionality on the performance of our model. The results on 4 datasets are shown in Figure 6.5. It can be observed that with a small dimension of embedding size, e.g. 16, our model achieved inferior performance on all the datasets. Thus, a small embedding size is not sufficient to express the latent features of users and items. By increasing the dimension of the embedding, the model's performance improves and approaches stability when the dimension reaches 128.

6.4.1.4 Impact of λ_1 and λ_2

In Eq. (6.16), λ_1 and λ_2 are hyper-parameters to control the weight of losses from individual views and contrastive losses among the views respectively. Properly controlling the weight of these two terms can enhance the performance of the model. In Figure 6.6, we show the impact of varying different values of λ_1 and λ_2 to the performance of our model on the datasets Yalp and ML-1M. When $\lambda_1 = 0$ or $\lambda_2 = 0$, the corresponding loss term is removed from our proposed model. As can be seen a moderate weight on the regularization terms enhances the performance of our model.

6.4.1.5 Adaptability to Different Encoder Networks

Table 6.11: Ablation study for different item graph construction methods. We compare our proposed item graph construction through time sequence, shown in (a); as well as the item graph construction through co-occurrence, shown in (b).

Dataset	ML-100k		Games		Steam		ML-1M	
Metric	HR@10	NDCG@10	HR@10	NDCG@10	HR@10	NDCG@10	HR@10	NDCG@10
(a) Time sequence	**0.0453**	**0.0230**	**0.2519**	**0.1352**	**0.1020**	**0.0536**	**0.0682**	**0.0360**
(b) Co-occurrence	0.0446	0.0221	0.2506	0.1345	0.0948	0.0483	0.0539	0.0287

Our proposed graph-based sequential recommendation model can not only be applied to transformers backbone but also able to enhance representation learning for other sequential networks such as RNNs. In Table 6.10, we show the adaptivity of our model to several commonly used sequential encoders including user gating networks (i.e. HGN), RNN module (i.e. GRU), and convolution networks (i.e. Caser). In this experiment, we replace the default transformer networks with the aforementioned networks, while keeping other parts the same as our proposed model. The results show that our graph-based multi-view model enhances the performance of these architectures. Specifically, the Caser model's performance increases from 0.028 to 0.0496 and from 0.0145 to 0.0286 on HR@10 in video games and CD datasets respectively.

6.4.1.6 Forms of Item Correlation Graphs

In Table 6.11 (b), we adopt another commonly used item-correlation graph: if two items are clicked by the same user, we add an edge to the item pair, and if the edge has been added, we add +1 to the edge's weight. After constructing the graph, we normalized the weights to make the adjacency matrix symmetrically column-stochastic, i.e., the sum of each column of the adjacency matrix is one. Since the adjacency matrix is symmetric, it is also row-stochastic. In the following steps, we use the normalized item-correlation graph to train our model, and the rest of the procedures follow our method in Section 6.2. Compared to Table 6.11 (a), which represents our default proposed graph construction method, we can recognize that constructing item-dependency graphs based on the time sequence of user's behaviors achieves higher performance, which also serves as a key aspect in the sequential recommendations.

To summarize, we mainly contribute the performance advantages of our model into four aspects,

- The transformer model can perfectly handle the item sequences and learn the item embeddings with respect to the user's trend of preference.
- The usage of item dependency sub-graphs and the hierarchical graph aggregation model can help to improve the representation capability and accuracy.

- The formation of multiple views and the application of mutual information maximization enhance the model's performance.
- The usage of Dirichlet sampling can mitigate the stiffness of the sub-graphs. By involving perturbations of the graph sampling, the model can learn a more robust estimation of the user's preference.

Together, as demonstrated in the ablation experiments, these components contribute to the model's overall effectiveness in addressing recommendation challenges and provide a powerful and efficient solution for predicting users' next preferred items in a general sequential recommendation setting.

6.5 Conclusions and Future Work

In this chapter, we propose a multi-view graph-based sequential recommendation model, in which we design the hierarchical graph aggregation networks to aggregate local information of items from the item dependency graph, and we apply the transformer model to process the sub-graph representations of each user-clicked item. Finally, we combine the representations from multiple views to predict the next preferred item by the user. In our model, we create Dirichlet weight sampling and mutual information maximization techniques to enhance our model's accuracy. Dirichlet weight sampling enables our model to dynamically sample multi-hop neighbours with low time and memory costs while preventing our model from overfitting on specific sets of high-weighted neighbours. From the time complexity perspective, we demonstrate that our model is suitable for modeling user's sequential behaviors with evidently low time cost, compared to applying other types of GNN-based user behavior modeling in sequential recommendations.

To evaluate our model, we conduct extensive experiments using five publicly available recommendation datasets and compare its performance against multiple state-of-the-art baselines. Our model consistently outperforms the competing baselines. Ablation studies further confirm the effectiveness of each component in our model. In future work, we plan to generalize our model for personalized recommendations that balance long-term and short-term user preferences, ensuring a more nuanced understanding of users' interests. Another promising area is to expand our model's applicability by incorporating general knowledge graphs, enabling the capture of a wider range of relationships and information for improved recommendations.

Chapter 7

Conclusion and Future Work

This thesis has demonstrated our ongoing efforts to address the remaining under-explored problems in graph representation and its applications via deep learning models. In Chapter 3, this thesis proposes a data augmentation method for graph representation learning through a meta-learning mechanism. Another aspect that significantly affects graph representation learning is the graph neural networks. In Chapter 4, this thesis presents a novel graph neural network that can capture graph signals in a multiresolution manner. Apart from the representation learning mechanism and the graph neural networks, this thesis turns our focus on the downstream tasks including graph generation (Chapter 5) and graph-based recommender systems (Chapter 6). In Chapter 5, this thesis presents a novel graph generation method via spectrum diffusion on graphs. In Chapter 6, this thesis presents a novel multi-view graph neural network for sequential recommendation. We hope that our research has advanced this field slightly. In the following sections, we will delve into other pertinent topics and outline our vision for potential future work.

Graph Meta-Contrast (GMeCo). We introduce a novel meta-learning framework for contrastive representation learning on graphs. This approach addresses the limitations of existing contrastive methods by using a meta-learner to generate augmented graphs adaptively. The meta-learner optimizes two key objectives: maximizing mutual information between the representations of augmented and input graphs, and ensuring the augmented graph is more similar to the input graph than any negative sample. Extensive experiments demonstrate its superior performance compared to state-of-the-art methods, emphasizing its effectiveness in robust and discriminative feature learning for graphs.

Multiresolution Meta-Framelet-based Graph Convolutional Network (MM-FGCN). In this work, we present a novel approach for graph representation learning. It introduces the Multiresolution Meta-Framelet-based Graph Convolutional Network (MM-FGCN), which enables adaptive multiresolution analysis of graphs. This model overcomes the limitations of fixed transforms by learning graph-specific transforms. The approach combines meta-learning with graph convolution networks, providing a dynamic

and flexible method for handling graph data across various scales. The experiments demonstrate superior performance in various graph learning tasks, showcasing the MM-FGCN's effectiveness and adaptability in capturing both micro and macro-level graph structures.

Graph Spectral Diffusion (GSDM). In this work, we present a novel approach to generating graph-structured data. It introduces the Graph Spectral Diffusion Model (GSDM), which improves upon existing models by leveraging low-rank diffusion Stochastic Differential Equations (SDEs) in graph spectrum space. This method addresses the limitations of full-rank diffusion SDEs, providing enhanced graph topology generation and significantly reducing computational load. The GSDM demonstrates superior performance in experiments, offering higher-quality graph generation and efficiency compared to baseline methods. We also include theoretical analyses to support the efficacy of GSDM in graph data generation.

Multi-View GNN-Transformers for sequential recommendation. In this work, we present an innovative framework for sequential recommendation systems. It introduces a novel architecture that combines Graph Neural Networks (GNNs) and Transformers to leverage both user-item interaction sequences and collaborative information among users. This approach addresses the limitations of traditional sequential recommendation models by incorporating a multi-view structure, enabling a more robust and accurate prediction of user preferences. We demonstrate the effectiveness our the model through extensive experiments, showing superior performance over existing methods. This research contributes to the field of recommender systems by offering a more comprehensive and dynamic model for understanding and predicting user behavior.

7.1 Possible Future Research Directions

This section presents some possible research directions for graph representation learning and some downstream tasks. This thesis aims to advance the methodologies and models in learning graph representation and the effective usage in the downstream tasks, thus it is necessary to address the future techniques and usage of graph-based models in various downstream tasks.

This thesis significantly contributes to graph representation learning and recommendation systems. It introduces innovative approaches, including spectral diffusion, meta-learning, and graph neural networks, to improve analysis and prediction in diverse applications. The research emphasizes adaptive, multi-resolution analysis and collaborative, sequential recommendation strategies, advancing beyond conventional methods. Future research should aim to tailor these models for specific uses, incorporate advanced technologies like specially designed deep graph neural networks, and examine their applicability in practical contexts. We summarize the future directions in three aspects: graph

representation learning, conditional graph generation models, and graph foundation models.

Future works on graph representation learning. Future research should prioritize the development of effective graph topology and node feature representation methods, recognizing the pivotal role of graph topology in defining graph characteristics. Graphs are generally classified into assortative or disassortative types based on node homophily, which measures the similarity of neighboring nodes. Assortative graphs have high homophily, with neighboring nodes often in the same class, while disassortative graphs have low homophily, with neighboring nodes typically in different classes. Traditional Graph Convolutional Networks (GCNs), based on low-pass filters, struggle to effectively capture features in disassortative graphs. This thesis introduces a multi-resolution model to address this issue. Future research could explore more sophisticated models that adeptly capture diverse information types. Another promising direction is designing adaptive graph architectures to automatically discern and leverage features in various graph types.

Future works on conditional graph generation. The conditional graph generation is required in various domains such as molecule modeling, drug discovery, social network analysis, etc. Future research can focus on advancing architectures to offer precise control over generated graphs, exploring multi-modal and dynamic graph generation, and addressing privacy concerns. Additionally, semi-supervised approaches and novel evaluation metrics can enhance model robustness and assessment. Transfer learning applications and real-world implementations, especially in domains like drug discovery and recommendation systems, are essential, as are efforts to make generated graphs interpretable and user-interactable, ultimately facilitating user-guided customization.

Future works on graph foundation models. The next frontier in graph-based research involves the development of graph foundation models, a concept inspired by the success of foundation models in fields like natural language processing and computer vision. These models would be pre-trained on extensive graph datasets, capturing a wide array of structural and relational patterns inherent in graph data. Future efforts should focus on creating these robust, versatile models capable of generalizing across various domains and tasks. Challenges to address include developing methods for effective pre-training, ensuring the models' adaptability to different types of graphs (e.g., assortative, disassortative), and refining them for specific downstream applications. Such models hold the potential to revolutionize how we approach complex graph-related problems, ranging from social network analysis to molecular structure prediction, by providing a comprehensive, pre-trained foundation that can be fine-tuned for specific tasks.

List of Publications

1. Zhanfeng Mo, **Tianze Luo**, Sinno Jialin Pan. "Graph Principal Flow Network for Conditional Graph Generation". Accepted by the ACM WebConf (WWW) (2024).

2. **Tianze Luo**, Zhanfeng Mo, Sinno Jialin Pan. "Learning Adaptive Multiresolution Transforms via Meta-Framelet-based Graph Convolutional Network". Accepted by the International Conference on Learning Representations (ICLR) (2024).

3. **Tianze Luo**, Zhanfeng Mo, Sinno Jialin Pan. "Fast Graph Generation via Spectral Diffusion". Accepted by IEEE Transactions on Pattern Analysis and Machine Intelligence (2023).

4. **Tianze Luo**, Zhanfeng Mo, Sinno Jialin Pan. "Conditional Graph Generation with Graph Principal Flow Network". International Conference on Machine Learning (ICML) 2023 Workshop on Structured Probabilistic Inference & Generative Modeling (2023).

5. **Tianze Luo**, Yong Liu, Sinno Jialin Pan. "Collaborative Sequential Recommendations via Multi-view GNN-Transformers". Under review at ACM Transactions on Information Systems (2023).

6. **Tianze Luo**, Qiuhao Zeng, Tianbo Li, and Sinno Jialin Pan, "Meta-contrast for Graph Representation Learning". Under review at IEEE Transactions on Pattern Analysis and Machine Intelligence (2023).

7. Quanyu Long, **Tianze Luo**, Wenya Wang, Sinno Jialin Pan. "Domain Confused Contrastive Learning for Unsupervised Domain Adaptation". Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL) (2022).

8. Tianbo Li, **Tianze Luo**, Yiping Ke, Sinno Jialin Pan. "Mitigating Performance Saturation in Neural Marked Point Processes: Architectures and Loss Functions." Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (2021).

References

[1] Bijaya Adhikari, Yao Zhang, Naren Ramakrishnan, and B Aditya Prakash. Sub2vec: Feature learning for subgraphs. In *PAKDD*, pages 170–182, 2018.

[2] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd international conference on World Wide Web*, pages 37–48, 2013.

[3] Brian D.O. Anderson. Reverse-time diffusion equation models. *Stochastic Processes and their Applications*, 12(3):313–326, 1982.

[4] László Babai and Ludik Kucera. Canonical labelling of graphs in linear average time. In *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 39–46. IEEE, 1979.

[5] Davide Bacciu, Alessio Conte, and Francesco Landolfi. Generalizing downsampling from regular data to graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 6718–6727, 2023.

[6] Muhammet Balcilar, Guillaume Renton, Pierre Héroux, Benoit Gaüzère, Sébastien Adam, and Paul Honeine. Analyzing the expressive power of graph neural networks in a spectral perspective. In *International Conference on Learning Representations*, 2021.

[7] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

[8] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7:2399–2434, 2006.

[9] Lorenz C Blum and Jean-Louis Reymond. 970 million druglike small molecules for virtual screening in the chemical universe database gdb-13. *Journal of the American Chemical Society*, 131(25):8732–8733, 2009.

[10] Deyu Bo, Xiao Wang, Chuan Shi, and Huawei Shen. Beyond low-frequency information in graph convolutional networks. *arXiv:2101.00797*, 2021.

[11] Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *International Conference on Data Mining*, 2005.

[12] Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *Fifth IEEE international conference on data mining (ICDM'05)*, pages 8–pp. IEEE, 2005.

[13] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations*, 2014.

[14] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pages 891–900, 2015.

[15] Heng Chang, Yu Rong, Tingyang Xu, Yatao Bian, Shiji Zhou, Xin Wang, Junzhou Huang, and Wenwu Zhu. Not all low-pass filters are robust in graph convolutional networks. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[16] Jianxin Chang, Chen Gao, Yu Zheng, Yiqun Hui, Yanan Niu, Yang Song, Depeng Jin, and Yong Li. Sequential recommendation with graph neural networks. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval*, pages 378–387, 2021.

[17] Fenxiao Chen, Yun-Cheng Wang, Bin Wang, and C-C Jay Kuo. Graph representation learning: a survey. *APSIPA Transactions on Signal and Information Processing*, 9, 2020.

[18] Huiyuan Chen, Lan Wang, Yusan Lin, Chin-Chia Michael Yeh, Fei Wang, and Hao Yang. Structured graph convolutional networks with stochastic masks for recommender systems. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 614–623, 2021.

[19] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv:1801.10247*, 2018.

[20] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International conference on machine learning*, pages 1725–1735. PMLR, 2020.

[21] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 257–266, 2019.

[22] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. *arXiv preprint arXiv:2006.07988*, 2020.

[23] Ashish Chiplunkar, Michael Kapralov, Sanjeev Khanna, Aida Mousavifar, and Yuval Peres. Testing graph clusterability: Algorithms and lower bounds. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 497–508. IEEE, 2018.

[24] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP*, pages 1724–1734, 2014.

[25] A Cohen, I Daubechies, B Jawerth, and P Vial. Multiresolution analysis, wavelets and fast algorithms on an interval. *Comptes rendus de l'Académie des sciences. Série 1, Mathématique*, 316(5):417–421, 1993.

[26] Gabriele Corso, Hannes Stärk, Bowen Jing, Regina Barzilay, and Tommi S. Jaakkola. Diffdock: Diffusion steps, twists, and turns for molecular docking. In *International Conference on Learning Representations*, 2023.

[27] Fabrizio Costa and Kurt De Grave. Fast neighborhood subgraph pairwise distance kernel. In *International Conference on Machine Learning*, 2010.

[28] Mark Craven, Andrew McCallum, Dan PiPasquo, Tom Mitchell, and Dayne Freitag. Learning to extract symbolic knowledge from the world wide web. Technical report, Carnegie-mellon univ pittsburgh pa school of computer Science, 1998.

[29] Valentin De Bortoli, Emile Mathieu, Michael Hutchinson, James Thornton, Yee Whye Teh, and Arnaud Doucet. Riemannian score-based generative modelling. *Advances in Neural Information Processing Systems*, 35:2406–2422, 2022.

[30] Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018.

[31] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852. 2016.

[32] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 3844–3852, Red Hook, NY, USA, 2016. Curran Associates Inc.

[33] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

[34] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, pages 4171–4186, 2019.

[35] Ming Ding, Jie Tang, and Jie Zhang. Semi-supervised learning on graphs with generative adversarial nets. In *Proceedings of the 27th ACM international conference on information and knowledge management*, pages 913–922, 2018.

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

[37] Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE international conference on computer vision*, pages 1422–1430, 2015.

[38] Jeff Donahue and Karen Simonyan. Large scale adversarial representation learning. *arXiv:1907.02544*, 2019.

[39] Bin Dong. Sparse representation on graphs by tight wavelet frames and applications. *Applied and Computational Harmonic Analysis*, 42(3):452–479, 2017.

[40] Monroe D Donsker and SR Srinivasa Varadhan. Asymptotic evaluation of certain markov process expectations for large time. *Communications on Pure and Applied Mathematics*, 28(1):1–47, 1975.

[41] Ziwei Fan, Zhiwei Liu, Yu Wang, Alice Wang, Zahra Nazari, Lei Zheng, Hao Peng, and Philip S Yu. Sequential recommendation via stochastic self-attention. *arXiv:2201.06035*, 2022.

[42] Hongchao Fang, Sicheng Wang, Meng Zhou, Jiayuan Ding, and Pengtao Xie. Cert: Contrastive self-supervised learning for language understanding. *arXiv:2005.12766*, 2020.

[43] Miroslav Fiedler. Laplacian of graphs and algebraic connectivity. *Banach Center Publications*, 1(25):57–70, 1989.

[44] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[45] Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. Protein interface prediction using graph convolutional networks. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[46] Hongyang Gao and Shuiwang Ji. Graph u-nets. In *international conference on machine learning*, pages 2083–2092. PMLR, 2019.

[47] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. Large-scale learnable graph convolutional networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1416–1424, 2018.

[48] Thomas Gärtner, Peter Flach, and Stefan Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Learning theory and kernel machines*, pages 129–143. Springer, 2003.

[49] Thomas Gärtner, Peter Flach, and Stefan Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Learning theory and kernel machines*, pages 129–143. 2003.

[50] Johannes Gasteiger, Stefan Weiß enberger, and Stephan Günnemann. Diffusion improves graph learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[51] Matan Gavish, Boaz Nadler, and Ronald R. Coifman. Multiscale wavelets on trees, graphs and high dimensional data: Theory and applications to semi supervised learning. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 367–374, Madison, WI, USA, 2010. Omnipress.

[52] Haoyu Geng, Chao Chen, Yixuan He, Gang Zeng, Zhaobing Han, Hua Chai, and Junchi Yan. Pyramid graph neural network: A graph sampling and filtering approach for multi-scale disentangled representations. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 518–530, 2023.

[53] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. *arXiv:1803.07728*, 2018.

[54] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pages 1263–1272, 2017.

[55] John M Giorgi, Osvald Nitski, Gary D Bader, and Bo Wang. Declutr: Deep contrastive learning for unsupervised textual representations. *arXiv:2006.03659*, 2020.

[56] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[57] Víctor González and Antonio Ortega. Multi-resolution spectral graph matching. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 2319–2323, 2019.

[58] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE, 2005.

[59] Marco Gori, Augusto Pucci, V Roma, and I Siena. Itemrank: A random-walk based scoring algorithm for recommender engines. In *IJCAI*, volume 7, pages 2766–2771, 2007.

[60] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[61] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *ACM SIGKDD*, pages 855–864, 2016.

[62] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *International Conference on Knowledge Discovery and Data Mining*, pages 855–864, 2016.

[63] Qingyu Guo, Fuzhen Zhuang, Chuan Qin, Hengshu Zhu, Xing Xie, Hui Xiong, and Qing He. A survey on knowledge graph-based recommender systems. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[64] Xiaojie Guo and Liang Zhao. A systematic survey on deep generative models for graph generation. *arXiv preprint arXiv:2007.06686*, 2020.

[65] Florentin Guth, Simon Coste, Valentin De Bortoli, and Stéphane Mallat. Wavelet score-based generative modeling. *ArXiv*, abs/2208.05003, 2022.

[66] Takuo Hamaguchi, Hidekazu Oiwa, Masashi Shimbo, and Yuji Matsumoto. Knowledge transfer for out-of-knowledge-base entities: A graph neural network approach. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, page 1802–1808. AAAI Press, 2017.

[67] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 1025–1035, Red Hook, NY, USA, 2017. Curran Associates Inc.

[68] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *IEEE Data Engineering Bulletin*, 2017.

[69] Bin Han, Zhenpeng Zhao, and Xiaosheng Zhuang. Directional tensor product complex tight framelets with low redundancy. *Applied and Computational Harmonic Analysis*, 41(2):603–637, 2016. Sparse Representations with Applications in Imaging Science, Data Analysis, and Beyond, Part II.

[70] Qi Hao, Tianze Luo, and Guangda Huzhang. Re-ranking with constraints on diversified exposures for homepage recommender system. *arXiv preprint arXiv:2112.07621*, 2021.

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

[72] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9729–9738, 2020.

[73] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

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

[75] Xiangnan He, Xiaoyu Du, Xiang Wang, Feng Tian, Jinhui Tang, and Tat-Seng Chua. Outer product-based neural collaborative filtering. *arXiv:1808.03912*, 2018.

[76] Xiangnan He, Zhankui He, Jingkuan Song, Zhenguang Liu, Yu-Gang Jiang, and Tat-Seng Chua. Nais: Neural attentive item similarity model for recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 30(12):2354–2366, 2018.

[77] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th International World Wide Web Conference*, pages 173–182, 2017.

[78] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015.

[79] Balázs Hidasi, Massimo Quadrana, Alexandros Karatzoglou, and Domonkos Tikk. Parallel recurrent neural network architectures for feature-rich session-based recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 241–248, 2016.

[80] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. In *International Conference on Learning Representations*, 2019.

[81] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey. Meta-learning in neural networks: A survey. *IEEE Transactions on Pattern Analysis amp; Machine Intelligence*, 44(09):5149–5169, sep 2022.

[82] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In Hugo Larochelle, Marc Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[83] Han Huang, Leilei Sun, Bowen Du, and Weifeng Lv. Conditional diffusion based on discrete graph structures for molecular graph generation. *arXiv preprint arXiv:2301.00427*, 2023.

[84] Jin Huang, Zhaochun Ren, Wayne Xin Zhao, Gaole He, Ji-Rong Wen, and Daxiang Dong. Taxonomy-aware multi-hop reasoning networks for sequential recommendation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 573–581, 2019.

[85] Vassilis N Ioannidis, Siheng Chen, and Georgios B Giannakis. Pruned graph scattering transforms. In *International Conference on Learning Representations*, 2020.

[86] John J Irwin, Teague Sterling, Michael M Mysinger, Erin S Bolstad, and Ryan G Coleman. Zinc: a free tool to discover chemistry for biology. *Journal of chemical information and modeling*, 52(7):1757–1768, 2012.

[87] Mingi Ji, Weonyoung Joo, Kyungwoo Song, Yoon-Yeong Kim, and Il-Chul Moon. Sequential recommendation with relation-aware kernelized self-attention. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 4304–4311, 2020.

[88] Bo Jiang, Doudou Lin, Jin Tang, and Bin Luo. Data representation and learning with graph diffusion-embedding networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 10414–10423, 2019.

[89] Bowen Jing, Gabriele Corso, Renato Berlinghieri, and Tommi Jaakkola. Subspace diffusion generative models. In *European Conference on Computer Vision*, pages 274–289. Springer, 2022.

[90] Bowen Jing, Gabriele Corso, Jeffrey Chang, Regina Barzilay, and Tommi S. Jaakkola. Torsional diffusion for molecular conformer generation. In *Advances in Neural Information Processing Systems*, 2022.

[91] Jaehyeong Jo, Seul Lee, and Sung Ju Hwang. Score-based generative modeling of graphs via the system of stochastic differential equations. *arXiv preprint arXiv:2202.02514*, 2022.

[92] Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In *2018 IEEE international conference on data mining (ICDM)*, pages 197–206. IEEE, 2018.

[93] Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. Marginalized kernels between labeled graphs. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pages 321–328, 2003.

[94] Jeroen Kazius, Ross McGuire, and Roberta Bursi. Derivation and validation of toxicophores for mutagenicity prediction. *Journal of Medicinal Chemistry*, 48(1):312–320, 2005.

[95] Shima Khoshraftar and Aijun An. A survey on graph representation learning methods. *arXiv preprint arXiv:2204.01855*, 2022.

[96] Diederik P Kingma and Jimmy Lei Ba. Adam: Amethod for stochastic optimization. In *International Conference on Learning Representations*, 2014.

[97] Diederik P. Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational diffusion models. *CoRR*, abs/2107.00630, 2021.

[98] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[99] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv:1611.07308*, 2016.

[100] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations*, ICLR '17, 2017.

[101] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. *arXiv:1911.05485*, 2019.

[102] Soheil Kolouri, Navid Naderializadeh, Gustavo K Rohde, and Heiko Hoffmann. Wasserstein embedding for graph learning. *arXiv preprint arXiv:2006.09430*, 2020.

[103] Risi Kondor and Horace Pan. The multiscale laplacian graph kernel. In *Advances in Neural Information Processing Systems*, pages 2990–2998. 2016.

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

[105] Walid Krichene and Steffen Rendle. On sampled metrics for item recommendation. In *KDD*, pages 1748–1757, 2020.

[106] Nils Kriege and Petra Mutzel. Subgraph matching kernels for attributed graphs. In *International conference on machine learning*, pages 291–298, 2012.

[107] Nils M Kriege, Pierre-Louis Giscard, and Richard Wilson. On valid optimal assignment kernels and applications to graph classification. In *Advances in Neural Information Processing Systems*, pages 1623–1631, 2016.

[108] Nils M Kriege, Fredrik D Johansson, and Christopher Morris. A survey on graph kernels. *Applied Network Science*, 5(1):1–42, 2020.

[109] H Kuhn and A Tucker. Nonlinear programming. In *In Proceedings of 2nd Berkeley symposium*, pages 481–492, 1951.

[110] Tuomas Kynkäänniemi, Tero Karras, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Improved precision and recall metric for assessing generative models. In *Advances in Neural Information Processing Systems*, 2019.

[111] Greg Landrum et al. Rdkit: Open-source cheminformatics software. 2016.

[112] Daniel D Lee and H Sebastian Seung. Algorithms for non-negative matrix factorization. In *Advances in neural information processing systems*, pages 556–562, 2001.

[113] Hayeon Lee, Eunyoung Hyung, and Sung Ju Hwang. Rapid neural architecture search by learning to generate graphs from datasets. *arXiv preprint arXiv:2107.00860*, 2021.

[114] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *International conference on machine learning*, 2019.

[115] Chenliang Li, Xichuan Niu, Xiangyang Luo, Zhenzhong Chen, and Cong Quan. A review-driven neural model for sequential recommendation. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, IJCAI'19, page 2866–2872. AAAI Press, 2019.

[116] Jia Li, Jiajin Li, Yang Liu, Jianwei Yu, Yueting Li, and Hong Cheng. Deconvolutional networks on graph data. *Advances in Neural Information Processing Systems*, 34:21019–21030, 2021.

[117] Jiacheng Li, Yujie Wang, and Julian McAuley. Time interval aware self-attention for sequential recommendation. In *WSDM*, pages 322–330, 2020.

[118] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. In *International Conference on Learning Representations*, 2016.

[119] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, 2018.

[120] Dawen Liang, Rahul G Krishnan, Matthew D Hoffman, and Tony Jebara. Variational autoencoders for collaborative filtering. In *Proceedings of the 2018 World Wide Web Conference*, pages 689–698, 2018.

[121] Xiaodan Liang, Xiaohui Shen, Jiashi Feng, Liang Lin, and Shuicheng Yan. Semantic object parsing with graph lstm. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 125–143. Springer, 2016.

[122] Renjie Liao, Zhizhen Zhao, Raquel Urtasun, and Richard Zemel. Lanczosnet: Multi-scale deep graph convolutional networks. In *International Conference on Learning Representations*, 2019.

[123] Guanyu Lin, Chen Gao, Yu Zheng, Jianxin Chang, Yanan Niu, Yang Song, Zhiheng Li, Depeng Jin, and Yong Li. Dual-interest factorization-heads attention for sequential recommendation. In *Proceedings of the ACM Web Conference 2023*, pages 917–927, 2023.

[124] Chaoyue Liu, Libin Zhu, and Mikhail Belkin. Toward a theory of optimization for over-parameterized systems of non-linear equations: the lessons of deep learning. *CoRR*, abs/2003.00307, 2020.

[125] Feng Liu, Qing Liu, Wei Guo, Huifeng Guo, Weiwen Liu, Ruiming Tang, Xutao Li, Yunming Ye, and Xiuqiang He. Inter-sequence enhanced framework for personalized sequential recommendation. *arXiv:2004.12118*, 2020.

[126] Huafeng Liu, Liping Jing, Jingxuan Wen, Pengyu Xu, Jiaqi Wang, Jian Yu, and Michael K Ng. Interpretable deep generative recommendation models. *J. Mach. Learn. Res.*, 22:202–1, 2021.

[127] Jenny Liu, Aviral Kumar, Jimmy Ba, Jamie Kiros, and Kevin Swersky. Graph normalizing flows. In *Advances in Neural Information Processing Systems*, 2019.

[128] Meng Liu, Zhengyang Wang, and Shuiwang Ji. Non-local graph neural networks. *IEEE transactions on pattern analysis and machine intelligence*, 44(12):10270–10276, 2021.

[129] Meng Liu, Zhengyang Wang, and Shuiwang Ji. Non-local graph neural networks. *IEEE transactions on pattern analysis and machine intelligence*, 44(12):10270–10276, 2021.

[130] Meng Liu, Zhengyang Wang, and Shuiwang Ji. Non-local graph neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(12):10270–10276, 2022.

[131] Meng Liu, Keqiang Yan, Bora Oztekin, and Shuiwang Ji. Graphebm: Molecular graph generation with energy-based models. *arXiv preprint arXiv:2102.00546*, 2021.

[132] Yang Liu and Maosong Sun. Contrastive unsupervised word alignment with non-local features. In *AAAI*, volume 29, 2015.

[133] Quanyu Long, Tianze Luo, Wenya Wang, and Sinno Jialin Pan. Domain confused contrastive learning for unsupervised domain adaptation. *arXiv preprint arXiv:2207.04564*, 2022.

[134] Qing Lu and Lise Getoor. Link-based classification. In *International conference on machine learning*, pages 496–503, 2003.

[135] Tianze Luo, Yong Liu, and Sinno Jialin Pan. Collaborative sequential recommendations via multi-view gnn-transformers. *ACM Transactions on Information Systems*, 2024.

[136] Tianze Luo, Zhanfeng Mo, and Sinno Jialin Pan. Conditional graph generation with graph principal flow network. In *ICML 2023 Workshop on Structured Probabilistic Inference and Generative Modeling*, 2023.

[137] Tianze Luo, Zhanfeng Mo, and Sinno Jialin Pan. Fast graph generation via spectral diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[138] Tianze Luo, Zhanfeng Mo, and Sinno Jialin Pan. Learning adaptive multiresolution transforms via meta-framelet-based graph convolutional network. In *The Twelfth International Conference on Learning Representations*, 2024.

[139] Youzhi Luo, Keqiang Yan, and Shuiwang Ji. Graphdf: A discrete flow model for molecular graph generation. In *International Conference on Machine Learning*, pages 7192–7203, 2021.

[140] Chen Ma, Peng Kang, and Xue Liu. Hierarchical gating networks for sequential recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 825–833, 2019.

[141] Tengfei Ma, Jie Chen, and Cao Xiao. Constrained generation of semantically valid graphs via regularizing variational autoencoders. In *Advances in Neural Information Processing Systems*, 2018.

[142] Yao Ma, Xiaorui Liu, Neil Shah, and Jiliang Tang. Is homophily a necessity for graph neural networks? In *International Conference on Learning Representations*, 2022.

[143] Stephane G Mallat. A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):674–693, 1989.

[144] Stephane G. Mallat. *A Theory for Multiresolution Signal Decomposition: The Wavelet Representation*, pages 494–513. Princeton University Press, Princeton, 2006.

[145] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. In *International Conference on Learning Representations*, 2019.

[146] Alessio Micheli. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511, 2009.

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

[148] Andriy Mnih and Russ R Salakhutdinov. Probabilistic matrix factorization. In *Advances in neural information processing systems*, pages 1257–1264, 2008.

[149] Zhanfeng Mo, Tianze Luo, and Sinno Jialin Pan. Graph principal flow network for conditional graph generation. In *Proceedings of the ACM on Web Conference 2024*, pages 768–779, 2024.

[150] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017.

[151] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. graph2vec: Learning distributed representations of graphs. *arXiv:1707.05005*, 2017.

[152] Alex Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. *CoRR*, abs/2102.09672, 2021.

[153] Chenhao Niu, Yang Song, Jiaming Song, Shengjia Zhao, Aditya Grover, and Stefano Ermon. Permutation invariant graph generation via score-based generative modeling. In *AISTATS*, pages 4474–4484, 2020.

[154] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-gan: Training generative neural samplers using variational divergence minimization. In *Advances in Neural Information Processing Systems*, pages 271–279, 2016.

[155] Hoang Nt and Takanori Maehara. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550*, 2019.

[156] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv:1807.03748*, 2018.

[157] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1105–1114, 2016.

[158] Zhiqiang Pan, Fei Cai, Wanyu Chen, Honghui Chen, and Maarten de Rijke. Star graph neural networks for session-based recommendation. In *CIKM*, pages 1195–1204, 2020.

[159] Sung-Jun Park, Dong-Kyu Chae, Hong-Kyun Bae, Sumin Park, and Sang-Wook Kim. Reinforcement learning over sentiment-augmented knowledge graphs towards accurate and explainable recommendation. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, pages 784–793, 2022.

[160] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. *arXiv preprint arXiv:2002.05287*, 2020.

[161] Nanyun Peng, Hoifung Poon, Chris Quirk, Kristina Toutanova, and Wen-tau Yih. Cross-sentence n-ary relation extraction with graph lstms. *Transactions of the Association for Computational Linguistics*, 5:101–115, 2017.

[162] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[163] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *KDD*, pages 701–710, 2014.

[164] Kristina Preuer, Philipp Renz, Thomas Unterthiner, Sepp Hochreiter, and Gunter Klambauer. Fréchet chemnet distance: a metric for generative models for molecules in drug discovery. *Journal of chemical information and modeling*, 58(9):1736–1741, 2018.

[165] Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. Gcc: Graph contrastive coding for graph neural network pre-training. In *KDD*, pages 1150–1160, 2020.

[166] Ruihong Qiu, Hongzhi Yin, Zi Huang, and Tong Chen. Gag: Global attributed graph neural network for streaming session-based recommendation. In *SIGIR*, 2020.

[167] Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole Von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1(1):1–7, 2014.

[168] Ruiyang Ren, Zhaoyang Liu, Yaliang Li, Wayne Xin Zhao, Hui Wang, Bolin Ding, and Ji-Rong Wen. Sequential recommendation with self-attentive multi-adversarial network. *arXiv:2005.10602*, 2020.

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

[170] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *WWW*, pages 811–820, 2010.

[171] Steffen Rendle, Walid Krichene, Li Zhang, and John Anderson. Neural collaborative filtering vs. matrix factorization revisited. *arXiv:2005.09683*, 2020.

[172] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding. *Journal of Complex Networks*, 9(2):cnab014, 2021.

[173] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4470–4479. PMLR, 10–15 Jul 2018.

[174] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Application of dimensionality reduction in recommender system-a case study. Technical report, Minnesota Univ Minneapolis Dept of Computer Science, 2000.

[175] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Incremental singular value decomposition algorithms for highly scalable recommender systems. In *Fifth international conference on computer and information science*, volume 1, pages 27–8. Citeseer, 2002.

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

[177] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93–93, 2008.

[178] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[179] Guy Shani and Asela Gunawardana. Evaluating recommendation systems. In *Recommender systems handbook*, pages 257–297. Springer, 2011.

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

[181] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In *AISTATS*, pages 488–495, 2009.

[182] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In *Artificial intelligence and statistics*, pages 488–495. PMLR, 2009.

[183] Chence Shi, Minkai Xu, Zhaocheng Zhu, Weinan Zhang, Ming Zhang, and Jian Tang. Graphaf: a flow-based autoregressive model for molecular graph generation. *arXiv preprint arXiv:2001.09382*, 2020.

[184] Jiaxin Shi, Chen Liang, Lei Hou, Juanzi Li, Zhiyuan Liu, and Hanwang Zhang. Deepchannel: Salience estimation by contrastive learning for extractive document summarization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6999–7006, 2019.

[185] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE signal processing magazine*, 30(3):83–98, 2013.

[186] Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *ICANN*, pages 412–422, 2018.

[187] Yang Song and Stefano Ermon. *Generative Modeling by Estimating Gradients of the Data Distribution*. Curran Associates Inc., Red Hook, NY, USA, 2019.

[188] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

[189] Alessandro Sperduti and Antonina Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.

[190] Elias M. Stein. *Harmonic Analysis: Real-Variable Methods, Orthogonality, and Oscillatory Integrals*. Princeton University Press, 1993.

[191] Fan-Yun Sun, Jordan Hoffman, Vikas Verma, and Jian Tang. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. In *International Conference on Learning Representations*, 2020.

[192] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *CIKM*, pages 1441–1450, 2019.

[193] Susheel Suresh, Vinith Budde, Jennifer Neville, Pan Li, and Jianzhu Ma. Breaking the limit of graph neural networks by improving the assortativity of graphs with local mixing patterns. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery amp; Data Mining*, KDD '21, page 1541–1551, New York, NY, USA, 2021. Association for Computing Machinery.

[194] Susheel Suresh, Pan Li, Cong Hao, and Jennifer Neville. Adversarial graph augmentation to improve graph contrastive learning. *Advances in Neural Information Processing Systems*, 34, 2021.

[195] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

[196] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *International Conference on World Wide Web*, pages 1067–1077, 2015.

[197] Jianheng Tang, Jiajin Li, Ziqi Gao, and Jia Li. Rethinking graph neural networks for anomaly detection. In *International Conference on Machine Learning*, pages 21076–21089. PMLR, 2022.

[198] Jiaxi Tang and Ke Wang. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 565–573, 2018.

[199] Mingyue Tang, Carl Yang, and Pan Li. Graph auto-encoder via neighborhood wasserstein reconstruction. *arXiv preprint arXiv:2202.09025*, 2022.

[200] Shantanu Thakoor, Corentin Tallec, Mohammad Gheshlaghi Azar, Mehdi Azabou, Eva L Dyer, Remi Munos, Petar Veličković, and Michal Valko. Large-scale representation learning on graphs via bootstrapping. In *International Conference on Learning Representations*, 2021.

[201] Kiran K Thekumparampil, Chong Wang, Sewoong Oh, and Li-Jia Li. Attention-based graph neural network for semi-supervised learning. arXiv:1803.03735, 2018.

[202] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive multiview coding. *arXiv:1906.05849*, 2019.

[203] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive multiview coding. *arXiv:1906.05849*, 2019.

[204] Yonglong Tian, Chen Sun, Ben Poole, Dilip Krishnan, Cordelia Schmid, and Phillip Isola. What makes for good views for contrastive learning. *arXiv:2005.10243*, 2020.

[205] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. Verse: Versatile graph embeddings from similarity measures. In *International Conference on World Wide Web*, page 539–548, 2018.

[206] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[207] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[208] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[209] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. *arXiv preprint arXiv:1809.10341*, 2018.

[210] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[211] Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. In *International Conference on Learning Representations*, 2019.

[212] Clement Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. Digress: Discrete denoising diffusion for graph generation. *arXiv preprint arXiv:2209.14734*, 2022.

[213] Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural Comput.*, 23(7):1661–1674, 2011.

[214] Nikil Wale, Ian A Watson, and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14(3):347–375, 2008.

[215] Hao Wang, Tong Xu, Qi Liu, Defu Lian, Enhong Chen, Dongfang Du, Han Wu, and Wen Su. Mcne: an end-to-end framework for learning multiple conditional network representations of social network. In *KDD*, pages 1064–1072, 2019.

[216] Hongwei Wang, Jialin Wang, Jia Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Wenjie Li, Xing Xie, and Minyi Guo. Learning graph representation with generative adversarial nets. *EEE Trans. Knowl. Data Eng.*, 33(8):3090–3103, 2019.

[217] Pengfei Wang, Yu Fan, Long Xia, Wayne Xin Zhao, ShaoZhang Niu, and Jimmy Huang. Kerl: A knowledge-guided reinforcement learning model for sequential recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 209–218, 2020.

[218] Tongzhou Wang and Phillip Isola. Understanding contrastive representation learning through alignment and uniformity on the hypersphere. In *International conference on machine learning*, pages 9929–9939, 2020.

[219] Wen Wang, Wei Zhang, Shukai Liu, Qi Liu, Bo Zhang, Leyu Lin, and Hongyuan Zha. Beyond clicks: Modeling multi-relational item graph for session-based target behavior prediction. In *The Web Conference*, 2020.

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

[221] Xiting Wang, Kunpeng Liu, Dongjie Wang, Le Wu, Yanjie Fu, and Xing Xie. Multi-level recommendation reasoning over knowledge graphs with reinforcement learning. In *Proceedings of the ACM Web Conference 2022*, pages 2098–2108, 2022.

[222] Ziyang Wang, Wei Wei, Gao Cong, Xiao-Li Li, Xian-Ling Mao, and Minghui Qiu. Global context enhanced graph neural networks for session-based recommendation. In *Proceedings of the 43rd ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 169–178, 2020.

[223] Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. Deep learning via semi-supervised embedding. In *International conference on machine learning*, pages 1168—-117, 2012.

[224] Kevin E. Wu, Kevin Kaichuang Yang, Rianne van den Berg, James Zou, Alex X. Lu, and Ava P. Amini. Protein structure generation via folding diffusion. *ArXiv*, abs/2209.15611, 2022.

[225] Liwei Wu, Shuqing Li, Cho-Jui Hsieh, and James Sharpnack. Sse-pt: Sequential recommendation via personalized transformer. In *Fourteenth ACM Conference on Recommender Systems*, pages 328–337, 2020.

[226] Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. Session-based recommendation with graph neural networks. In *AAAI*, pages 346–353, 2019.

[227] Yongji Wu, Defu Lian, Yiheng Xu, Le Wu, and Enhong Chen. Graph convolutional networks with markov random field reasoning for social spammer detection. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(01):1054–1061, Apr. 2020.

[228] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[229] Xin Xia, Hongzhi Yin, Junliang Yu, Qinyong Wang, Lizhen Cui, and Xiangliang Zhang. Self-supervised hypergraph convolutional networks for session-based recommendation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 4503–4511, 2021.

[230] Bingbing Xu, Huawei Shen, Qi Cao, Yunqi Qiu, and Xueqi Cheng. Graph wavelet neural network. In *International Conference on Learning Representations*, 2019.

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

[232] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International conference on machine learning*, pages 5453–5462, 2018.

[233] Kun Xu, Lingfei Wu, Zhiguo Wang, Yansong Feng, Michael Witbrock, and Vadim Sheinin. Graph2seq: Graph to sequence learning with attention-based neural networks. *arXiv preprint arXiv:1804.00823*, 2018.

[234] Nuo Xu, Pinghui Wang, Long Chen, Jing Tao, and Junzhou Zhao. Mr-gnn: Multi-resolution and dual graph neural network for predicting structured entity interactions. *arXiv preprint arXiv:1905.09558*, 2019.

[235] Pinar Yanardag and S.V.N. Vishwana. Deep graph kernels. In *International Conference on Knowledge Discovery and Data Mining*, pages 1365–1374, 2015.

[236] Yiding Yang, Zunlei Feng, Mingli Song, and Xinchao Wang. Factorizable graph convolutional networks. In *Advances in Neural Information Processing Systems*, pages 20286–20296, 2020.

[237] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pages 40–48, 2016.

[238] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International Conference on Machine Learning*, pages 5708–5717, 2018.

[239] Yuning You, Tianlong Chen, Yang Shen, and Zhangyang Wang. Graph contrastive learning automated. In *International conference on machine learning*, pages 12121–12132, 2021.

[240] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. In *Advances in Neural Information Processing Systems*, 2020.

[241] Yuning You, Tianlong Chen, Zhangyang Wang, and Yang Shen. Bringing your own view: Graph contrastive learning without prefabricated data augmentations. In *WSDM*, pages 1300–1309, 2022.

[242] Feng Yu, Yanqiao Zhu, Qiang Liu, Shu Wu, Liang Wang, and Tieniu Tan. Tagnn: Target attentive graph neural networks for session-based recommendation. In *SIGIR*, pages 1921–1924, 2020.

[243] Honglin Yuan, Xiaoyi Gu, Rongjie Lai, and Zaiwen Wen. Global optimization with orthogonality constraints via stochastic diffusion on manifold. *Journal of Scientific Computing*, 80(2):1139–1170, 2019.

[244] Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM Transactions On Algorithms*, 1(1):2–13, 2005.

[245] Chengxi Zang and Fei Wang. Moflow: an invertible flow model for generating molecular graphs. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 617–626, 2020.

[246] Qiuhao Zeng, Tianze Luo, and Boyu Wang. Domain-augmented domain adaptation. *arXiv preprint arXiv:2202.10000*, 2022.

[247] Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit-Yan Yeung. GaAN: Gated attention networks for learning on large and spatiotemporal graphs. In *UAI*, pages 339–349, 2018.

[248] Jiawei Zhang, Haopeng Zhang, Congying Xia, and Li Sun. Graph-bert: Only attention is needed for learning graph representations. *arXiv preprint arXiv:2001.05140*, 2020.

[249] Liheng Zhang, Guo-Jun Qi, Liqiang Wang, and Jiebo Luo. Aet vs. aed: Unsupervised representation learning by auto-encoding transformations rather than data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2547–2555, 2019.

[250] Richard Zhang, Phillip Isola, and Alexei A Efros. Split-brain autoencoders: Unsupervised learning by cross-channel prediction. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1058–1067, 2017.

[251] Sheng Zhang, Rui Song, and Wenbin Lu. Graphcgan: Convolutional graph neural network with generative adversarial networks. *Openreview Preprint*, 2020.

[252] Kai Zhao, Yukun Zheng, Tao Zhuang, Xiang Li, and Xiaoyi Zeng. Joint learning of e-commerce search and recommendation with a unified graph neural network. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, pages 1461–1469, 2022.

[253] Xuebin Zheng, Bingxin Zhou, Junbin Gao, Yuguang Wang, Pietro Lió, Ming Li, and Guido Montufar. How framelets enhance graph neural networks. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 12761–12771. PMLR, 18–24 Jul 2021.

[254] Xuebin Zheng, Bingxin Zhou, Yu Guang Wang, and Xiaosheng Zhuang. Decimated framelet system on graphs and fast G-framelet transforms. *Journal of Machine Learning Research*, 2021.

[255] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. Deep interest evolution network for click-through rate prediction. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 5941–5948, 2019.

[256] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. Deep interest evolution network for click-through rate prediction. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 5941–5948, 2019.

[257] Guorui Zhou, Xiaoqiang Zhu, Chengru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. Deep interest network for click-through rate prediction. In *KDD*, pages 1059–1068, 2018.

[258] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

[259] Meiqi Zhu, Xiao Wang, Chuan Shi, Houye Ji, and Peng Cui. Interpreting and unifying graph neural networks with an optimization framework. In *Proceedings of the Web Conference 2021*, pages 1215–1226, 2021.

[260] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *International conference on machine learning*, pages 912–919, 2003.

[261] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Deep graph contrastive representation learning. *arXiv:2006.04131*, 2020.

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