OPTIMAL GRAPH LEARNING

BY

ZHE XU

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois Urbana-Champaign, 2025

Urbana, Illinois

Doctoral Committee:

Professor Hanghang Tong, Chair
Professor Arindam Banerjee
Dr. Yuzhong Chen, Visa Research
Professor Jiawei Han

ABSTRACT

The past decades have seen significant advancements in graph machine learning, with numerous sophisticated models and algorithms crafted for a variety of learning tasks, including ranking, classification, regression, and anomaly detection. Generally, most existing works focus on addressing the question: *given a graph, what is the best way to mine it?*

Despite their remarkable achievements, little attention is paid to the graph data itself, which could be noisy, huge, and imbalanced at every stage of the data collection process. In this thesis, our focus is on the relatively unexplored realm of graph data, intending to enhance various downstream graph machine learning tasks. We term this line of research *optimal graph learning*, aiming to identify the most effective graph data to improve efficiency, effectiveness, and expressiveness.

However, some unique challenges arise. First (*formulation*), it is not clear how to formulate data optimization in a data-driven way, especially considering that the downstream tasks can be versatile. Second (*volume*), the sheer volume of graph datasets can result in significant time and space complexity for underlying optimization solutions. Third (*pattern*), capturing various essential graph patterns at different granularities presents a challenge.

This thesis introduces our progress towards the optimal graph learning problem. Concretely, we categorize our work into three directions: **graph refinement**, **graph augmentation**, and **graph distillation**. For graph refinement, we developed (1) a pure data-driven solution named GaSoliNe against noisy data and (2) Stager, a solution tailored for addressing imbalanced data. For graph augmentation, we developed three augmentation solutions: (1) ALT, enhancing broad models' performance on graphs with arbitrary heterophily, (2) DisCo, which can generate realistic graphs based on the training graphs, and (3) AuGLM , which incorporates the graph structure into the textual input so that the language models can successfully handle the node classification task. For graph distillation, we developed (1) a bilevel optimization-based solution named KiDD to shrink the size of given graphs and, meanwhile, preserve the utility of training data and (2) graph rationale discovery framework named FIG, which can find the critical subgraph in every given graph to enhance the performance of graph-level performance.

Collectively, these contributions establish foundational progress toward data-centric graph machine learning and demonstrate the value of optimizing graph data itself to improve downstream task performance.

To my parents, Yiying He and Shunping Xu.

ACKNOWLEDGMENTS

This journey is long, and I would definitely like to express my gratitude to many.

I want to express my heartfelt gratitude to my advisor, Prof. Hanghang Tong, for his intelligence, deep expertise, humility, and patience. Although I am not a top-talented student, he has consistently placed his trust in me, invested time in my growth, and provided invaluable opportunities. His research vision and academic rigor have profoundly shaped my development, and I am sure they will continue to guide and inspire my future career.

I want to thank Prof. Jingrui He, the director of our sister lab. She and Prof. Tong make the iDEA-iSAIL joint lab a true home for all the members.

I want to thank my thesis committee members, Prof. Arindam Banerjee, Dr. Yuzhong Chen, Prof. Jiawei Han, and my advisor, Prof. Hanghang Tong. Their mentorship, insightful feedback, and guidance have been essential to the development and successful completion of this thesis.

I am grateful to all the lab members who shared this PhD journey with me. They are my important collaborators and brainstorming partners. They are: Xing Su, Liangyue Li, Chen Chen, Arun Reddy Nelakurthi, Shweta Jain, Yao Zhou, Dawei Zhou, Si Zhang, Xu Liu, Boxin Du, Yuheng Zhang, Shengyu Feng, Yian Wang, Derek Wang, Qinghai Zhou, Jian Kang, Ishika Agarwal, Blaine Hill, Yunyong Ko, Jun Wu, Lihui Liu, Dongqi Fu, Yikun Ban, Yuchen Yan, Lecheng Zheng, Jun-Gi Jang, Baoyu Jing, Wenxuan Bao, Ziwei Wu, Tianxin Wei, Yunzhe Qi, Eunice Chan, Xinrui He, Xinyu He, Isaac Joy, Zhining Liu, Hyunsik Yoo, Xiao Lin, Zihao Li, Ruizhong Qiu, Zhichen Zeng, Mengting Ai, Sirui Chen, Gaotang Li, Ting-Wei Li, Jiaru Zou, Xuying Ning, Xue Hu, Yaojing Wang, Ziye Zhu, Yu Wang, Yancheng Wang, Hansheng Ren, Hao Wang, Haonan Wang, Haoran Li, Zihao Wang, Haobo Xu, Tianwen Chen.

I want to extend my gratitude to my academic partners, collaborators, and mentors beyond the iDEA-iSAIL lab. I deeply value and appreciate our collaborations. They are Dr. Yinglong Xia, Dr. Liang Xiong, Dr. Jiejun Xu, Prof. Kaize Ding, Prof. Yu-Xiong Wang, Prof. Huan Liu, Dr. Yuzhong Chen, Dr. Yuhang Wu, Dr. Menghai Pan, Dr. Hao Yang, Dr. Huiyuan Chen, Dr. Mahashweta Das, Dr. Qing Chen, Nan Chen, Wei Shuai, Guande Wu, Dr. Nan Cao, Dr. Yada Zhu, Dr. Kommy Weldemariam, Prof. Tarek F. Abdelzaher, Prof. Lei Ying, Dr. Xiran Fan, Dr. Chin-Chia Michael Yeh, Dr. Vivian Lai, Dr. Yan Zheng, Dr. Minghua Xu, Dr. Kaveh Hassani, Dr. Hanqing Zeng, Dr. Michihiro Yasunaga, Dr. Limei Wang, Dr. Ning Yao, Dr. Bo Long, Dr. Xiaoting Li.

I want to thank Prof. Yanghua Xiao and the KW@FUDAN lab. Prof. Xiao is my advisor for undergraduate research. He, along with senior members of the lab, offered hands-on guidance when I was first introduced to academic research. Without their support, I might not have had the opportunity to pursue a PhD.

Lastly, I would like to express my deepest gratitude to my family. My parents, Yiying He and Shunping Xu, have unconditionally supported the PhD study of their only child, thousands of miles away, for many years. Their selfless support and enduring belief in me will always be the origin of my warmth. My gratitude certainly goes to my wife, Yunyun He, who has stood by me with patience and understanding throughout the most intense and demanding phases of my academic journey. She willingly embraced a life devoid of entertainment during every 'paper-submission', 'rebuttal', 'job-finding', and 'pre-presentation' moment. Her unwavering trust and support have been my driving force toward greater and more ambitious goals. I am also thankful for the smallest member of our family, Remy, our golden hammie, whose companionship during countless late-night hours, as he ran on his wheel beside me, brought me a quiet but resilient power.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION 1
1.1 Motivation 1
1.2 Overview of Thesis Research 2

CHAPTER 2 LITERATURE REVIEW 4
2.1 Graph Refinement 4
2.2 Graph Augmentation 5
2.3 Graph Distillation 6

CHAPTER 3 GRAPH REFINEMENT 7
3.1 Graph Sanitation with Application to Node Classification 7
3.2 Generalized Few-Shot Node Classification 26

CHAPTER 4 GRAPH AUGMENTATION 46
4.1 Node Classification Beyond Homophily: Towards a General Solution 46
4.2 Discrete-State Continuous-Time Diffusion for Graph Generation 65
4.3 How to Make LMs Strong Node Classifiers? 95

CHAPTER 5 GRAPH DISTILLATION 115
5.1 Kernel Ridge Regression-based Graph Dataset Distillation 115
5.2 Fine-Grained Graph Rationalization 135

CHAPTER 6 CONCLUSION AND FUTURE WORK 153
6.1 Conclusion 153
6.2 Future Work 153

REFERENCES 155

CHAPTER 1: INTRODUCTION

1.1 MOTIVATION

Graph mining has emerged as a cornerstone in a plethora of real-world applications, including social network mining [1, 2, 3, 4, 5, 6, 7], brain connectivity analysis [8], computational epidemiology [9], and financial analysis [10, 11]. The majority of existing works strive to address the fundamental question: *given* a graph, what is the best model and/or algorithm to mine it? Notable examples include (1) PageRank [12] and its variants [13, 14, 15, 16, 17, 18], which measure node importance and proximity based on multiple weighted paths, (2) spectral clustering [19], which partitions nodes into distinct groups by minimizing inter-cluster connectivity and maximizing intra-cluster connectivity, and (3) graph neural networks (GNNs) [20, 21, 22, 23, 24, 25, 26, 27], which learn node representations by aggregating information from neighborhood nodes. All these approaches necessitate a given graph, encompassing its topology and/or associated attribute information, as input for the corresponding mining model.

Despite their remarkable achievements, several fundamental questions remain largely unanswered. For instance, where does the input graph originate? To what extent does the quality of the given graph influence the effectiveness of the corresponding mining model? Can we distill a smaller graph dataset without compromising performance on downstream tasks? How can we enhance a broad spectrum of graph machine learning models to accommodate varying heterophily patterns [28, 29] within a given graph? All the above data-centric questions can be summarized as a general but succinct one: *what is the optimal graph for a specific downstream task?* We name this research problem as *optimal graph learning*.

However, some unique challenges arise for such a data-centric research problem. First (*formulation*), it is not clear how to formulate the optimization objective of data in a data-driven way, especially considering the versatile downstream tasks. For example, how can we ensure the performance of a specific graph machine learning model can be improved after the given graph is optimized? How can we ensure a family of graph machine learning models (e.g., off-the-shelf graph neural networks) can uniformly benefit from the optimized graphs? Second (*volume*), the given graph dataset can be huge and lead to high time and space complexity of the underlying optimization solutions. For example, real-world graph data [30] contains hundreds of thousands of graphs and nodes; thus, trivially treating the entire graph data as an optimization variable is resource-intensive [31]. In addition, real-world graph data typically exhibits sparse connectivity, yet many optimization-based solutions yield densely

connected optimal graphs. Third (*pattern*), the graph patterns are critical at different granularities, e.g., locally every node has different homophily/heterophily tendencies against its 1-hop neighbors [32], which might affect the performance of downstream graph machine learning models. In addition, for graph-level tasks, such as graph generation and graph classification [33, 34], some substructures are critical for the property of the whole graph, e.g., functional groups in molecule graphs [35, 36]. It remains unclear how to capture, preserve, and emphasize these patterns within the optimal graph learning framework.

1.2 OVERVIEW OF THESIS RESEARCH

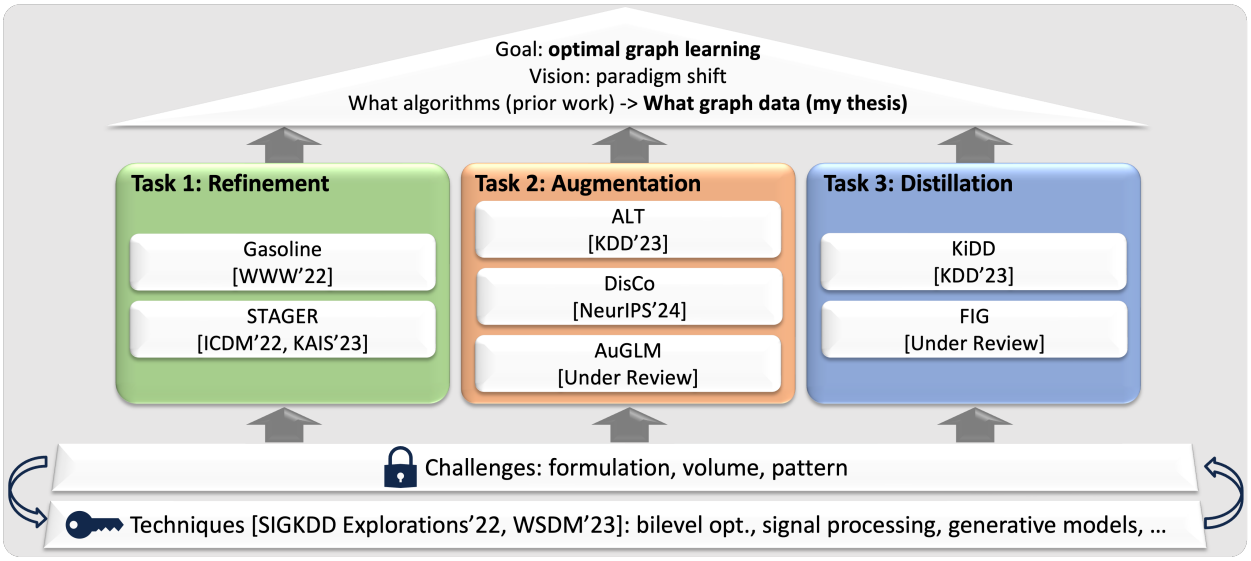


Figure 1.1: An overview of this thesis, optimal graph learning.

To address these challenges, my Ph.D. thesis proposes novel algorithms and solutions for a broad spectrum of graph machine learning tasks, including node classification, graph classification, graph regression, and graph generation. Specifically, we delve into three distinct sub-tasks: **graph refinement**, **graph augmentation**, and **graph distillation**. Figure 1.1 shows an overview of the challenges and tasks of this thesis proposal.

- *Graph Refinement.* Graph refinement refers to fixing the graphs with inherent flaws, such as noise and label imbalance. This problem has been studied for a long time. For example, NeuralSparse [37] and PTDNet [38] model the edge-dropping probability using categorical and Bernoulli distributions, respectively, and drop edges based on sampling strategies. In our completed work, we have proposed a flexible data-driven framework named GASOLINE [31] which can improve various graph mining

tasks against noisy and adversarially attacked graphs. We also have developed a meta-learning and uncertainty-based solution STAGER [39] to address the label imbalance problem in node classification tasks.

- *Graph Augmentation.* Graph augmentation enriches the given graph data to improve the downstream models' performance. It has been widely used in various tasks, including graph classification, node classification, and contrastive learning. For example, it [40, 41] can be used to generate more training samples by mixing the features and labels of existing training graphs. For this task, we have developed 3 data-driven solutions named ALT [42], DisCo [43], and AUGLM . ALT decomposes the given graph into two graphs with supplementary homophily/heterophily properties to improve various backbone graph neural networks' performance on the node classification tasks. DisCo is a discrete-state continuous-time graph generation framework that produces realistic samples efficiently. AUGLM is a language model-based node classification model whose core idea is to enhance the textual data via input graphs to empower the language model for node classification.

- *Graph Distillation.* Graph distillation is a set of tasks shrinking the size of a given graph dataset, either in terms of the number of nodes or the number of graphs. This direction has been studied for both node-level datasets [44] and graph-level datasets [45] through the bilevel optimization objective. Some of them [45] apply aggressive approximate solutions to mitigate the computation overhead. In this direction, we have completed a work named KIDD [46], which provides an exact and efficient solution to distill graph-level datasets. We also developed FIG [47], which tackles the problem from a rationalization perspective by identifying task-relevant subgraphs to improve graph classification and regression performance maximally.

CHAPTER 2: LITERATURE REVIEW

2.1 GRAPH REFINEMENT

Due to various factors such as fake connections [48], over-personalized users [49], and construction heuristics, the given graph structure may not be optimal for downstream graph learning tasks [50]. A subset of solutions that focus on the graph topology is termed *graph rewiring*.

Several methods exist to rewire given graphs using various node similarity metrics. In many cases, these metrics are learned from the given graph topology. For instance, approaches like GAUG [51] and IDGL [52] train edge predictors based on learned node embeddings. Furthermore, from an optimization perspective, it is feasible to directly incorporate the graph data (e.g., adjacency matrix) itself as part of the optimization variables. Based on that, the *graph rewiring* process is essentially guided by the optimization objective. A notable example is TO-GNN [53], where loss functions include smoothness-related regularizations and gradient descent-based graph updates. Additionally, instead of optimizing the graph itself, an intriguing idea is to optimize graph-related distributions (e.g., graph generation and edge-dropping distributions). After that, *graph rewiring* can be conducted by sampling from those distributions. An exemplary work is LDS [54], which assumes that each edge is sampled from an independent Bernoulli distribution. Other noteworthy works in this domain include Bayesian-GCNN [55], GEN [56], NeuralSparse [37], and PTDNet [38].

In comparison to structure refinement, research on graph feature refinement has gained less attention. In general, most of the work is developed based on *feature rewriting*. For instance, AirGNN [57] regularizes the l_{21} norm between input node features and convoluted node features to increase model tolerance against abnormal features. To handle missing node features, a special case of suboptimal initial node features, feature propagation [58] diffuses features from observed nodes to neighbors with missing features based on the heat diffusion equation, essentially imputing missing node features with aggregated features from neighboring nodes. GCNMF [59] explicitly formulates missing node features using Gaussian mixture models whose parameters are inferred from downstream tasks. Efforts like SAT [60] reconstruct missing features through the feature distribution inferred from the topology distribution. To handle missing features in heterogeneous information networks, HGNN-AC [61] imputes missing features from neighbor nodes' topology-based node embeddings, while HGCA [62] designs a feature augmenter trained to maximize agreement between augmented node embeddings and actual node embeddings.

2.2 GRAPH AUGMENTATION

A typical application of graph augmentation is addressing the issue of data scarcity [63]. One effective solution is to leverage unlabeled data to augment the limited labeled data. Following the concept of pseudo-labeling, self-training [64] assigns labels to unlabeled data based on a teacher model trained with limited labeled data, becoming a prevailing approach for semi-supervised node classification in scenarios with limited training data. Among these methods, Li et al. [65] first combine Graph Convolutional Networks (GCNs) with self-training to expand supervision signals. CGCN [66] generates pseudo labels by combining a variational graph auto-encoder with Gaussian mixture models. Furthermore, M3S [67] proposes multi-stage self-training and utilizes a clustering method to eliminate potentially incorrect pseudo labels. Similar concepts are also explored in [68]. Additionally, recent research [69, 70] employs label propagation as the teacher model to generate pseudo labels that encode valuable global structural knowledge.

Similar to self-training, co-training [71] has been investigated for augmenting training sets with unlabeled data. It learns two classifiers using initial labeled data on two views and allows them to label each other's unlabeled data to augment the training data. Li et al. [72] develop a novel multi-view semi-supervised learning method, Co-GCN, based on feature masking, which unifies GCN and co-training into one framework.

Another approach to obtaining additional training examples [43] is to use an interpolation-based data augmentation strategy, such as Mixup [73, 74, 75], to generate synthetic training examples (i.e., node insertion) through feature mixing and label mixing. While graphs have arbitrary structures, unlike images or natural sentences, identifying meaningful connections between original and synthetic nodes remains a challenging task. Additionally, due to the cascading effect of graph data, simply adding an edge can dramatically alter semantic meanings. To address these challenges, Manifold Mixup [76] is applied to graph data interpolation. For instance, GraphMix [77] trains a multi-layer perceptron (MLP) jointly with GNNs via parameter sharing, where the MLP is learned based on Manifold Mixup and pseudo-labeling, effectively training GNNs for semi-supervised node classification. Similarly, Wang et al. [78] leverage the idea of Manifold Mixup and interpolate input features of both nodes and graphs in the embedding space. These methods use a straightforward approach to circumvent dealing with arbitrary structures by mixing graph representations learned from GNNs.

For input-level graph data interpolation, ifMixup [79] targets the Manifold Intrusion issue by first interpolating node features and edges based on feature mixing and graph generation. Graph Transplant [40] utilizes graph rewiring to mix dissimilar-structured graphs by re-

placing the destination subgraph with the source subgraph while preserving local structure. G-Mixup [80] estimates the graphon of each class, performs mixup between graphons, and conducts graph generation to generate interpolated graphs, improving the generalizability and robustness of GNNs for semi-supervised graph classification.

2.3 GRAPH DISTILLATION

A popular line of graph distillation is to extract a small graph from a large graph, named *graph coarsening* [81]. Its core idea of graph coarsening is to minimize a 'quantity of interest' between the input graph $\mathcal{G}$ and augmented graph $\tilde{\mathcal{G}}$. For example, Jin et al. [82] propose to minimize the spectral distance between $\mathcal{G}$ and $\tilde{\mathcal{G}}$. GOREN [83] aims to maintain comparable Laplace operators between $\mathcal{G}$ and $\tilde{\mathcal{G}}$.

Beyond that, a set of tasks named dataset distillation [84, 85] has been widely studied recently. Their goal is to find a small but informative dataset that preserves the training utility from the given large dataset. Its application on graphs attracts decent attention but has not been thoroughly studied. The optimization objective of two typical examples, GCOND [44] and DosCond [45] is the difference of training gradients on the given graph $\mathcal{G}$ and the target small graph $\tilde{\mathcal{G}}$. Our complete work, KIDD [46]stands as one of the pioneering endeavors in this area.

Beyond the above works, some methods that focus on interpretability and out-of-distribution generation will highlight a critical subgraph, e.g., graph rationalization [35]. These endeavors can also be regarded as solutions within the realm of graph distillation.

CHAPTER 3: GRAPH REFINEMENT

3.1 GRAPH SANITATION WITH APPLICATION TO NODE CLASSIFICATION

3.1.1 Introduction

Graph mining has become the cornerstone in a wealth of real-world applications, such as social network mining [1, 2, 86], brain connectivity analysis [8], computational epidemiology [9], and financial analysis [10, 11]. For the vast majority of existing works, they essentially aim to answer the following question, that is, *given* a graph, what is the best model and/or algorithm to mine it? To name a few, PageRank [12] and its variants [13, 14, 15, 16] measure the node importance and node proximity based on multiple weighted paths; spectral clustering [19] minimizes inter-cluster connectivity and maximizes the intra-cluster connectivity to partition nodes into different groups; graph neural networks (GNNs) [20, 21, 22, 23] learn representation of nodes by aggregating information from the neighborhood. In all these works and many more, they require a *given* graph, including its topology and/or the associated attribute information, as part of the input of the corresponding mining model.

Despite tremendous success, some fundamental questions largely remain open, e.g., where does the input graph come from at the first place? To what extent does the quality of the given graph impact the effectiveness of the corresponding graph mining model? In response, we introduce the *graph sanitation* problem, which aims to improve the initially provided graph for a given graph mining model, to boost its performance maximally. The rationality is as follows. In many existing graph mining works, the initially provided graph is typically constructed manually based on some heuristics. The graph construction is often treated as a pre-processing step, without the consideration of the specific mining task. Furthermore, the initially constructed graph may be subject to various forms of contamination, including missing information, noise, and even adversarial attacks [87, 88]. This suggests that there may be an under-explored space for improving mining performance by learning a 'better' graph as input to the corresponding task.

There are several existing lines of work for modifying graphs. For example, network imputation [89, 90, 91] and knowledge graph completion [92, 93, 94, 95, 96, 97] problems focus on restoring missing links in a partially observed graph; connectivity optimization [98] and computational immunization [99] problems manipulate the graph connectivity in a desired way by changing the underlying topology; robust GNNs [100, 101, 102] utilize empirical properties of a benign graph to remove or assign lower weights to the poisoned graph elements

(e.g., contaminated edges).

The graph sanitation problem introduced in this paper is related to but bears a subtle difference from the existing work in the following sense. Most, if not all, of these existing works for modifying graphs assume the initially provided graph is impaired or perturbed in a specific way, e.g., due to missing links, noise, or adversarial attacks. Some existing works further impose certain assumptions on specific graph modification algorithms, such as the low-rank assumption underlying many network imputation methods, the types of attacks, and/or the empirical properties of the benign graph (e.g., topology sparsity, feature smoothness) behind some robust graph neural networks (GNNs). In contrast, the proposed graph sanitation problem does not make any such assumptions; instead, it pursues a different design principle. That is, we aim to let the performance of the downstream data mining task, measured on a validation set, dictate how we should optimally modify the initially provided graph. This is crucial, as it not only ensures that the modified graph will directly and maximally improve the mining performance, but also lends itself to being applied to a variety of graph mining tasks.

Formally, we formulate the graph sanitation problem as a generic bilevel optimization problem, where the lower-level optimization problem corresponds to the specific mining task, and the upper-level optimization problem encodes the supervision to modify the provided graph and maximize the improvement in mining performance. Based on this, we instantiate a bilevel optimization problem using semi-supervised node classification with GNNs, where the lower-level objective function represents the cross-entropy classification loss over the training data, and the upper-level objective function represents the loss over the validation data, utilizing the mining model trained from the lower-level optimization problem. We propose an effective solver (GaSoliNe) which adopts an efficient approximation of hyper-gradient to guide the modification over the given graph. We carefully design the hyper-gradient aggregation mechanism to avoid potential bias from a specific dataset split by aggregating the hyper-gradient from different folds of data. GaSoliNe is versatile, and is equipped with multiple variants, such as discretized vs. continuous modification, modifying graph topology vs. feature. Comprehensive experiments demonstrate that (1) GaSoliNe is broadly applicable to benefit different downstream node classifiers together with flexible choices of variants and modification strategies, (2) GaSoliNe can significantly boost downstream classifiers on both the original and contaminated graphs in various perturbation scenarios and can work hand-in-hand with existing robust GNNs methods. For instance, in Table 3.1, the proposed GaSoliNe significantly boosts GAT [21], SVD [100], and RGCN [103].

Table 3.1: Node classification accuracy (mean±std) boosting of existing defense methods on poisoned graphs (25% edges perturbed by metattack [104]) by the proposed GaSoliNe.

Data	With GaSoliNe?	GAT	SVD	RGCN
Cora	N	48.8±0.2	60.3±0.8	50.6±0.8
	Y	63.7±0.6	79.7±0.6	62.6±0.6
Citeseer	N	62.4±0.7	49.5±0.8	55.5±1.4
	Y	69.7±0.2	76.5±0.6	66.1±0.8
Polblogs	N	48.2±6.6	79.1±2.4	50.8±0.9
	Y	70.8±0.6	89.2±0.7	67.7±0.3

Table 3.2: Instantiations of the graph sanitation problem over various mining tasks

Tasks	**Personalized PageRank** [107, 108]	**Spectral Clustering** [19, 109]	**Node Classification** [110]
$\mathcal{L}_{\texttt{lower}}$	$\min_{\mathbf{r}}\ q\mathbf{r}'(\mathbf{I}-\bar{\mathbf{A}})\mathbf{r}+(1-q)\|\mathbf{r}-\mathbf{e}\|^2$	$\min_{\mathbf{u}}\ \mathbf{u}'\mathbf{L}\mathbf{u}$ s.t. $\mathbf{u}'\mathbf{D}\mathbf{u}=1,\ \mathbf{D}\mathbf{u}\perp\mathbf{1}$	$\min_{\theta}\ -\sum_{i\in\mathcal{T}}\sum_{j=1}^{c} y_{ij}\ln\hat{y}_{ij}$
$\mathcal{L}_{\texttt{upper}}$	$\min_{\mathbf{A}}\ \sum_{x\in\mathcal{P},y\in\mathcal{N}}(1+\exp\left(\mathbf{r}^*[x]-\mathbf{r}^*[y]\right)/w)^{-1}$	$\min_{\mathbf{A}}\ -\mathbf{u}^{*\prime}\mathbf{Q}\mathbf{u}^*$	$\min_{G}\ -\sum_{i\in\mathcal{V}}\sum_{j=1}^{c} y_{ij}\ln\hat{y}_{ij}$
$\mathcal{T}$	none	none	training set $\mathcal{T}$
$\mathcal{Y}_{\texttt{train}}$	none	none	labels of training set $\mathcal{Y}_{\texttt{train}}$
$\mathcal{V}$	positive node set $\mathcal{P}$, negative node set $\mathcal{N}$	'must-link' set $\mathcal{M}$, 'cannot-link' set $\mathcal{C}$	validation set $\mathcal{V}$
$\mathcal{Y}_{\texttt{valid}}$	none	none	labels of validation set $\mathcal{Y}_{\texttt{valid}}$
Remarks	normalized adjacency matrix $\bar{\mathbf{A}}$, damping factor q, width parameter w, preference vector $\mathbf{e}$	Laplacian matrix $\mathbf{L}$, degree matrix $\mathbf{D}$, link constraints matrix $\mathbf{Q}$	number of classes c, predicted probability $\hat{y}_{ij}$, ground truth label y_{ij}

3.1.2 Problem Definition

A - Optimization-Based Graph Mining Models. For many graph mining models, they can be formulated from the optimization perspective [105, 106] with a general goal to find an optimal solution θ^* so that a task-specific loss $\mathcal{L}(G, \theta, \mathcal{T}, \mathcal{Y}_{\texttt{train}})$ is minimized. Here, $\mathcal{T}$ and $\mathcal{Y}_{\texttt{train}}$ are the training set and the associated ground truth (e.g., class labels for the classification task), which would be absent for the unsupervised graph mining tasks (e.g., clustering, ranking). We give three concrete examples next.

Example #1: personalized PageRank [13, 111] is a fundamental ranking model. When the adjacency matrix of the underlying graph is symmetrically normalized, the ranking vector $\mathbf{r}$ can be obtained as:

$$\mathbf{r}^* = \arg\min_{\mathbf{r}}\ q\mathbf{r}'\left(\mathbf{I}-\bar{\mathbf{A}}\right)\mathbf{r}+(1-q)\left\|\mathbf{r}-\mathbf{e}\right\|^2, \tag{3.1}$$

where $\bar{\mathbf{A}}$ is the symmetrically normalized adjacency matrix; $q \in (0, 1]$ is the damping factor; $\mathbf{e}$ is the preference vector; the ranking vector $\mathbf{r}^*$ is the solution of the ranking model (i.e.,

$\theta^* = \mathbf{r}^*$).

Example #2: spectral clustering [19] is a classic graph clustering model aiming to minimize the normalized cut between clusters:

$$\mathbf{u}^* = \arg\min_{\mathbf{u}} \ \mathbf{u}'\mathbf{L}\mathbf{u} \quad \text{s.t. } \mathbf{u}'\mathbf{D}\mathbf{u} = 1, \ \mathbf{D}\mathbf{u} \perp \mathbf{1}_\mathbf{u}, \tag{3.2}$$

where $\mathbf{L}$ is the Laplacian matrix, $\mathbf{D}$ is the diagonal degree matrix (i.e., $\mathbf{D}[i,i] = \sum_j \mathbf{A}[i,j]$), $\mathbf{1}_u$ is an all-one vector with the same size as $\mathbf{u}$; the model solution θ^* is the cluster indicator vector $\mathbf{u}^*$.

Example #3: node classification aims to construct a classification model based on the graph topology $\mathbf{A}$ and feature $\mathbf{X}$. A typical loss for node classification is cross-entropy (CE) over the training set:

$$\theta^* = \arg\min_{\theta} \ -\sum_{i \in \mathcal{T}} \sum_{j=1}^{c} y_{ij} \ln \hat{y}_{ij}, \tag{3.3}$$

where c is the number of classes, y_{ij} is the ground truth indicating if node i belongs to class j, $\mathcal{T}$ is the training set, $\hat{y}_{ij} = f(G,\theta)[i,j]$ is the predicted probability that node i belongs to class j by a classifier $f(G,\theta)$ parameterized by θ. For example, classifier $f(G,\theta)$ can be a GNN whose trained model parameters form the solution θ^*.

Remarks. Both the standard personalized PageRank and spectral clustering are unsupervised. Therefore, the training set $\mathcal{T}$ and its supervision $\mathcal{Y}_{\texttt{train}}$ are absent in the corresponding loss functions (i.e., Eq. (3.1) and (3.2), respectively). Nonetheless, personalized PageRank and spectral clustering have been generalized to incorporate some forms of supervision further, as we will show next.

B - Graph Sanitation: Formulation and Instantiations. Given an initial graph G and an optimization-based graph mining model $\mathcal{L}(G, \theta, \mathcal{T}, \mathcal{Y}_{\texttt{train}})$, we aim to learn a modified graph $\tilde{G}$ to boost the performance of the corresponding mining model. We name it as *graph sanitation problem*. The basic idea is to let the mining performance on a validation set $\mathcal{V}$ guide the modification process. Formally, the graph sanitation problem is defined as follows:

Problem 3.1. Graph Sanitation Problem

Given: (1) a graph $G = \{\mathbf{A}, \mathbf{X}\}$, (2) a mining task $\mathcal{L}(G, \theta, \mathcal{T}, \mathcal{Y}_{\texttt{train}})$, (3) a validation set $\mathcal{V}$ and its supervision $\mathcal{Y}_{\texttt{valid}}$, and (4) the sanitation budget B;

Find: A modified graph $\tilde{G} = \{\tilde{\mathbf{A}}, \tilde{\mathbf{X}}\}$ to boost the graph mining performance.

We formulate Problem 3.1 as a bilevel optimization problem:

$$\tilde{G} = \arg\min_{G} \ \mathcal{L}_{\texttt{upper}}\left(G, \theta^*, \mathcal{V}, \mathcal{Y}_{\texttt{valid}}\right) \tag{3.4}$$

$$\text{s.t. } \theta^* = \arg\min \mathcal{L}_{\texttt{lower}}\left(G, \theta, \mathcal{T}, \mathcal{Y}_{\texttt{train}}\right), \ D\left(\tilde{G}, G\right) \leq B \tag{3.5}$$

where the lower-level optimization is to train the model θ^* based on the training set $\mathcal{T}$; the upper-level optimization aims to optimize the performance of the trained model θ^* on the validation set $\mathcal{V}$, and there is no overlap between $\mathcal{T}$ and $\mathcal{V}$; the distance function D measures the distance between two graphs. In this paper, we instantiate $D(\tilde{G}, G)$ as $\left\|\tilde{\mathbf{A}} - \mathbf{A}\right\|_{1,1}$ or $\left\|\tilde{\mathbf{X}} - \mathbf{X}\right\|_{1,1}$ based on the modification scenarios. Notice that the loss function at the upper level $\mathcal{L}_{\texttt{upper}}$ might be different from the one at the lower level $\mathcal{L}_{\texttt{lower}}$. For example, $\mathcal{L}_{\texttt{lower}}$ for both personalized PageRank (Eq. (3.1)) and spectral clustering (Eq. (3.2)) does not involve any supervision. However, $\mathcal{L}_{\texttt{upper}}$ for both models are designed to measure the performance on a validation set with supervision and, therefore, should be different from $\mathcal{L}_{\texttt{lower}}$. We elaborate on this next.

The proposed bilevel optimization problem in Eq. (3.4) is quite general. In principle, it is applicable to *any* graph model with differentiable $\mathcal{L}_{\texttt{upper}}$ and $\mathcal{L}_{\texttt{lower}}$. We give its instantiations with the three aforementioned mining tasks and summarize them in Table 3.2.

Instantiation #1: supervised PageRank. The original personalized PageRank [13] has been generalized to encode pair-wised ranking preference [107, 108]. For graph sanitation with supervised PageRank, the training set and its supervision is absent, and the lower-level loss $\mathcal{L}_{\texttt{lower}}$ is given in Eq. (3.1). The validation set $\mathcal{V}$ consists of a positive node set $\mathcal{P}$ and a negative node set $\mathcal{N}$. The supervision of the upper-level problem is that ranking scores of nodes from $\mathcal{P}$ should be higher than that from $\mathcal{N}$, i.e., $\mathbf{r}[x] > \mathbf{r}[y], \forall x \in \mathcal{P}, \forall y \in \mathcal{N}$. Several choices for the upper-level loss $\mathcal{L}_{\texttt{upper}}$ exist. For example, we can use Wilcoxon-Mann-Whitney loss [112]:

$$\min_{\mathbf{A}} \sum_{x \in \mathcal{P}, y \in \mathcal{N}} \left(1 + \exp\left(\mathbf{r}^*[x] - \mathbf{r}^*[y]\right)/w\right)^{-1} \tag{3.6}$$

where w is the width parameter. It is worth mentioning that Eq. (3.6) only modifies graph topology $\mathbf{A}$. Although Eq. (3.6) does not contain variable $\mathbf{A}$, $\mathbf{r}^*$ is determined by $\mathbf{A}$ through the lower-level problem.

Instantiation #2: supervised spectral clustering. A typical way to encode supervision in spectral clustering is via 'must-link' and 'cannot-link' [109, 113]. For graph sanitation with supervised spectral clustering, the training set together with its supervision is absent, and

the lower-level loss $\mathcal{L}_{\texttt{lower}}$ is given in Eq. (3.2). The validation set $\mathcal{V}$ contains a 'must-link' set $\mathcal{M}$ and a 'cannot-link' set $\mathcal{C}$. For the upper-level loss, the idea is to encourage nodes from must-link set $\mathcal{M}$ to be grouped in the same cluster and, in the meanwhile, push nodes from cannot-link set $\mathcal{C}$ to be in different clusters. To be specific, $\mathcal{L}_{\texttt{upper}}$ can be instantiated as follows.

$$\min_{\mathbf{A}} \quad -\mathbf{u}^{*\prime}\mathbf{Q}\mathbf{u}^{*}, \tag{3.7}$$

where $\mathbf{Q}$ encodes the 'must-link' and 'cannot-link', that is, $\mathbf{Q}[i,j] = 1$ if $(i,j) \in \mathcal{M}$, $\mathbf{Q}[i,j] = -1$ if $(i,j) \in \mathcal{C}$, and $\mathbf{Q}[i,j] = 0$ otherwise. This instantiation only modifies the graph topology $\mathbf{A}$.

Instantiation #3: semi-supervised node classification. For graph sanitation with semi-supervised node classification, its lower-level optimization problem is given in Eq. (3.3). We have cross-entropy loss over validation set $\mathcal{V}$ as the upper-level problem:

$$\min_{G} \quad \mathcal{L}_{\texttt{CE}}\left(G, \theta^{*}, \mathcal{V}, \mathcal{Y}_{\texttt{valid}}\right) = -\sum_{i \in \mathcal{V}} \sum_{j=1}^{c} y_{ij} \ln \hat{y}_{ij}. \tag{3.8}$$

Notice that $\mathcal{T} \cap \mathcal{V} = \emptyset$. If both the topology $\mathbf{A}$ and node feature $\mathbf{X}$ are used for classification, then they both can be modified in this instantiation.

Remarks. If the initially given graph G is poisoned by adversarial attackers [104, 114], the graph sanitation problem with semi-supervised node classification can also be used as a defense strategy. However, it bears an important difference from the existing robust GNNs [100, 101, 102] as it does not assume the given graph G is poisoned or any specific way by which it is poisoned. Therefore, the graph sanitation problem in this scenario can boost the performance under a wide range of attacking scenarios (e.g., non-poisoned graphs, lightly-poisoned graphs, and heavily-poisoned graphs) and has the potential to work hand-in-hand with existing robust GNNs model. The following section proposes an effective algorithm for the graph sanitation problem with semi-supervised node classification.

3.1.3 Proposed Algorithms: GaSoliNe

We focus on graph sanitation problem in the context of semi-supervised node classification and propose an effective solver named GaSoliNe. The general workflow is as follows. First, we solve the lower-level problem (Eq. (3.3)) and obtain a solution θ^{*}. Then we compute the hyper-gradient of the upper-level loss function (Eq. (3.8)) w.r.t. the graph G to solve the upper-level optimization problem. Recall that a classifier $f(\theta)$ is needed to provide

the predicted labels (in both the lower-level and upper-level problems) and we refer to this classifier as the *backbone classifier*. Finally we test the performance of another classifier over the modified graph on the test set $\mathcal{W}$ and this classifier is named as the *downstream classifier*. We will introduce our proposed solution GaSoliNe in four parts: (A) hyper-gradient computation, (B) hyper-gradient aggregation, (C) hyper-gradient-guided modification, and (D) low-rank speedup.

A - Hyper-Gradient Computation. Eq. (3.4) and its corresponding instantiations Eqs. (3.3)(3.8) fall into the family of bilevel optimization problem where the lower-level problem is to optimize θ via minimizing the loss over the training set $\{\mathcal{T}, \mathcal{Y}_{\texttt{train}}\}$ given G, and the upper-level problem is to optimize G via minimizing the loss over $\{\mathcal{V}, \mathcal{Y}_{\texttt{valid}}\}$. We compute gradient w.r.t. the upper-level problem and view the lower-level problem as a dynamic system:

$$\theta^{t+1} = \Theta^{t+1}(G, \theta^t, \mathcal{T}, \mathcal{Y}_{\texttt{train}}), \quad \theta^1 = \Theta^1(G, \mathcal{T}, \mathcal{Y}_{\texttt{train}}), \tag{3.9}$$

where Θ^1 is the initialization of θ and Θ^{t+1} ($t \neq 0$) is the updating formula which can be instantiated as an optimizer over the lower-level objective function on the training set (Eq. (3.3)). For the hyper-gradient of the upper-level problem $\nabla_G \mathcal{L}$, we assume that the system converges in T iterations (i.e., $\theta^* = \theta^T$). Then we unroll the iterative solution of the lower-level problem to obtain the hyper-gradient $\nabla_G \mathcal{L}$ by the chain rule as follows [115] where $A_t = \nabla_{\theta^t} \theta^{t+1}$, $B_t = \nabla_G \theta^{t+1}$. For brevity, we abbreviate cross-entropy loss over the validation set $\mathcal{L}_{\texttt{CE}}(G, \theta^T, \mathcal{V}, \mathcal{Y}_{\texttt{valid}})$ as $\mathcal{L}_{\texttt{valid}}(\theta^T)$.

$$\nabla_G \mathcal{L} = \nabla_G \mathcal{L}_{\texttt{valid}}\left(\theta^T\right) + \sum_{t=0}^{T-2} B_t A_{t+1} \dots A_{T-1} \nabla_{\theta^T} \mathcal{L}_{\texttt{valid}}\left(\theta^T\right) \tag{3.10}$$

Our final goal is to improve the performance of converged downstream classifiers. Hence, T is set as a relatively large value (e.g., 200) to ensure the hyper-gradient is computed over a converged classifier. To balance the effectiveness and the efficiency, we adopt the truncated hyper-gradient [116] w.r.t. G and rewrite the second part of Eq. (3.10) as $\sum_{t=P}^{T-2} B_t A_{t+1} \dots A_{T-1} \nabla_{\theta^T} \mathcal{L}_{\texttt{valid}}(\theta^T)$, where P denotes the truncating iteration. To further speed up, we adopt a first-order approximation [104, 117] and $\nabla_G \mathcal{L}$ can be computed as:

$$\nabla_G \mathcal{L} = \sum_{t=P}^{T} \nabla_G \mathcal{L}_{\texttt{valid}}\left(\theta^t\right). \tag{3.11}$$

where the updating trajectory of θ^t is the same as Eq. (3.9). If the initially-provided graph

G is undirected, it indicates that $\mathbf{A} = \mathbf{A}'$. Hence, when we compute the hyper-gradient w.r.t. the undirected graph topology $\mathbf{A}$, we need to calibrate the partial derivative into the derivative [105] and update the hyper-gradient as follows:

$$\nabla_{\mathbf{A}}\mathcal{L} \leftarrow \nabla_{\mathbf{A}}\mathcal{L} + (\nabla_{\mathbf{A}}\mathcal{L})' - \operatorname{diag}(\nabla_{\mathbf{A}}\mathcal{L}). \tag{3.12}$$

For the hyper-gradient w.r.t. feature $\mathbf{X}$ and directed graph topology $\mathbf{A}$ ($\mathbf{A} \neq \mathbf{A}'$), the above calibration process is not needed.

B - Hyper-Gradient Aggregation. To ensure the quality of graph sanitation without introducing bias from a specific dataset split, we adopt K-fold split with similar settings as cross-validation [49]. Specifically, during the training phase, we split all the labeled nodes $\mathcal{Z}$ into K folds and alternatively select one of them as $\mathcal{V}$ (with labels $\mathcal{Y}_{\texttt{valid}}$) and the others as $\mathcal{T}$ (with labels $\mathcal{Y}_{\texttt{train}}$). In total, there are K sets of training/validation splits. With the k-th dataset split, by Eq. (3.11), we obtain the hyper-gradient $\nabla_G^k\mathcal{L}$. For the hyper-gradient $\{\nabla_G^1\mathcal{L}, \ldots, \nabla_G^K\mathcal{L}\}$ from the K sets of training/validation split, we sum them up as the aggregated hyper-gradient: $\nabla_G = \sum_k \nabla_G^k\mathcal{L}$.

C - Hyper-Gradient-Guided Modification. To modify the graph based on ∇_G, we provide two variants, *discretized modification* and *continuous modification*. The discretized modification can work with binary inputs such as adjacency matrices of unweighted graphs and binary feature matrices. The continuous modification is suitable for both continuous and binary inputs. For clarity of explanation, we replace the G with the adjacency matrix $\mathbf{A}$ as an example of the topology modification. Generalizing that to the feature modification with feature matrix $\mathbf{X}$ is straightforward.

The discretized modification is flipping B entries in $\mathbf{A}$ whose indices are also the indices of the top-B entries in a hyper-gradient-based score matrix $\mathbf{S}$:

$$\mathbf{S} = (-\nabla_{\mathbf{A}}) \odot (\mathbf{1} - 2\mathbf{A}), \tag{3.13}$$

where $\odot$ denotes Hadamard product, $\mathbf{1}$ is an all-one matrix. This score matrix is composed of 'preference' (i.e., $-\nabla_{\mathbf{A}}$) and 'modifiability' (i.e., $(\mathbf{1} - 2\mathbf{A})$). Only high 'preference' and 'modifiability' entries are assigned with high scores. For example, large positive $(-\nabla_{\mathbf{A}})[i, j]$ indicates strong preference of adding an edge between the i-th and j-th nodes based on the hyper-gradient and if the i-th and j-th nodes are not linked (i.e., $\mathbf{A}[i, j] = 0$), $(-\nabla_{\mathbf{A}})[i, j]$ and $(\mathbf{1} - 2\mathbf{A})[i, j]$ share the same sign which results in a large $\mathbf{S}[i, j]$.

The continuous modification is gradient descent with a budget-adaptive learning rate:

$$\mathbf{A} \leftarrow \mathbf{A} - \frac{B}{\mathbf{1}^\top |\nabla_{\mathbf{A}}| \, \mathbf{1}} \cdot \nabla_{\mathbf{A}}. \tag{3.14}$$

The learning rate is based on the ratio of the modification budget B to the sum of absolute values of the gradient matrix. In implementation, for both modification methods, we set the budget in every iteration to b and update the graph in multiple steps until the total budget B is exhausted. Algorithm 3.1 summarizes our methods. In addition, in our experiments, the B for topology and feature ($B_{\texttt{topo}}$ and $B_{\texttt{fea}}$) are set separately since the modification cost on different elements of a graph may not be comparable.

Algorithm 3.1: GaSoliNe

Input : a graph G, the labeled nodes $\mathcal{Z}$, total budget and budget in every step B and b, number of fold K, truncating and converging iterations T and P;
Output: the modified graph $\tilde{G}$;

initialization: split the labeled nodes $\mathcal{Z}$ and their labels into K folds: $\mathcal{Z} = \{\mathcal{Z}^1, \ldots, \mathcal{Z}^K\}$, $\mathcal{Y}_{\texttt{labeled}} = \{\mathcal{Y}^1, \ldots, \mathcal{Y}^K\}$; $\tilde{G} \leftarrow G$; cumulative budget $\delta \leftarrow 0$;
while $\delta < B$ **do**
  for *k=1 to K* **do**
    $\mathcal{V} \leftarrow \mathcal{Z}^k$, $\mathcal{T} \leftarrow \mathcal{Z} \backslash \mathcal{Z}^k$, $\mathcal{Y}_{\texttt{valid}} \leftarrow \mathcal{Y}^k$, $\mathcal{Y}_{\texttt{train}} \leftarrow \mathcal{Y}_{\texttt{labeled}} \backslash \mathcal{Y}^k$, $\nabla^k_{\tilde{G}} \mathcal{L} \leftarrow 0$;
    for $t = 1$ *to* T **do**
      update θ^t to θ^{t+1} by Eq. (3.9);
      if $t > P$ **then**
        compute $\nabla^k_{\tilde{G}} \mathcal{L}_{\texttt{valid}}$ given $\{\tilde{G}, \theta^{t+1}, \mathcal{V}, \mathcal{Y}_{\texttt{valid}}\}$;
        $\nabla^k_{\tilde{G}} \mathcal{L} \leftarrow \nabla^k_{\tilde{G}} \mathcal{L} + \nabla^k_{\tilde{G}} \mathcal{L}_{\texttt{valid}}$
      end
    end
  end
  calibrate $\{\nabla^k_{\tilde{\mathbf{A}}} \mathcal{L}\}$ by Eq. (3.12) (if needed);
  sum $\{\nabla^k_{\tilde{G}} \mathcal{L}\}$ into $\nabla_{\tilde{G}}$, $\delta \leftarrow \delta + b$;
  update $\tilde{G}$ based on the guide of score matrix $\mathbf{S}$ by Eq. (3.13) (discretized modification) or by Eq. (3.14) (continuous modification) with budget b
end

D - Speedup and Scale-up. The core operation of our proposed GaSoliNe is to compute hyper-gradient w.r.t. the graph components (i.e., $\mathbf{A}$ and $\mathbf{X}$) which leads into a gradient matrix (e.g. $\nabla_{\mathbf{A}} \mathcal{L}$). In many real-world scenarios (e.g., malfunctions of certain nodes, targeted adversarial attacks), perturbations are often around a small set of nodes, which leads

to low-rank perturbation matrices. Hence, for topology modification, we propose to decompose the incremental matrix (i.e., $\Delta\mathbf{A}$) into its low-rank representation (i.e., $\Delta\mathbf{A} = \mathbf{U}\mathbf{V}'$), and compute the hyper-gradient with respect to the low-rank matrices instead, which can significantly speed up and scale up the computation. Recall that the low-rank assumption is only held for the incremental matrix but for the modified graph (i.e., $\tilde{\mathbf{A}}$), it is not limited to low-rank. Mathematically, the low-rank modification can be represented as:

$$\tilde{\mathbf{A}} = \mathbf{A} + \Delta\mathbf{A} = \mathbf{A} + \mathbf{U}\mathbf{V}', \tag{3.15}$$

where $\mathbf{U}$, $\mathbf{V} \in \mathbb{R}^{n\times r}$, and r is the rank of $\Delta\mathbf{A}$. Hence, by substituting $\mathbf{A}$ with $\mathbf{A} + \mathbf{U}\mathbf{V}'$ in Eq. (3.8) (i.e., $G = \{\mathbf{A}+\mathbf{U}\mathbf{V}', \mathbf{X}\}$) and changing the optimization variable from $\mathbf{A}$ into $\mathbf{U}$ and $\mathbf{V}$, we can obtain hyper-gradient with respect to $\mathbf{U}$ and $\mathbf{V}$ (i.e., $\nabla_{\mathbf{U}}\mathcal{L}$ and $\nabla_{\mathbf{V}}\mathcal{L}$) in the same manner as Eq.(3.11). By aggregating the hyper-gradients from different training/validation splits as we introduced in Sec. 3.1.3-B, we obtain aggregated hyper-gradients $\nabla\mathbf{U}$ and $\nabla\mathbf{V}$. Any gradient descent-based method can then be used to update $\mathbf{U}$ and $\mathbf{V}$.

In this way, we can significantly reduce the complexity of time and space, which is summarized in the following lemma. Notice that n, m, d are the number of nodes, the number of edges, and the feature dimension, respectively, and we have $d \ll n$ and $m \ll n^2$. As a comparison, the time complexity of computing $\nabla_{\mathbf{A}}\mathcal{L}$ is $O(n^2 d)$ and the space complexity of computing $\nabla_{\mathbf{A}}\mathcal{L}$ is $O(n^2)$. Hence, this low-rank method is much more efficient in both time and space.

Lemma 3.1. For computing $\nabla_{\mathbf{U}}\mathcal{L}$ and $\nabla_{\mathbf{V}}\mathcal{L}$, the time complexity is $O(nd^2 + md)$ and the space complexity is $O(m + nd)$.

Proof. For typical matrix multiplication-based GNNs (e.g., [20]), their propagation formula can be represented as $\mathbf{X} \leftarrow \sigma(\mathbf{A}\mathbf{X}\mathbf{W})$ (or even simplified by removing the nonlinear activation function σ and feature transformation matrix $\mathbf{W}$ between several layers [22, 23]). If we do not consider the gradient across the model parameter (i.e., $\mathbf{W}$) updating trajectory (i.e., first order approximation), and assume that our GNN contains only one layer, the hyper-gradient concerning the vector $\mathbf{U}$ can be computed as follows,

$$\nabla_{\mathbf{U}}\mathcal{L} = \left[\frac{\partial\mathcal{L}}{\partial\sigma\left(\left(\mathbf{A} + \mathbf{U}\mathbf{V}'\right)\mathbf{X}\mathbf{W}\right)} \odot \sigma'\left(\left(\mathbf{A} + \mathbf{U}\mathbf{V}'\right)\mathbf{X}\mathbf{W}\right)\right]\mathbf{W}'\mathbf{X}'\mathbf{V}. \tag{3.16}$$

The computation of $(\mathbf{A} + \mathbf{U}\mathbf{V}')\mathbf{X}\mathbf{W}$ can be rewritten as $\mathbf{A}\mathbf{X}\mathbf{W} + \mathbf{U}\mathbf{V}'\mathbf{X}\mathbf{W}$. Note that $\mathbf{A}$ is a sparse matrix and the space cost is $O(m)$ for computing $\mathbf{A}\mathbf{X}\mathbf{W}$. The space cost is $O(nd)$ for $\mathbf{U}\mathbf{V}'\mathbf{X}\mathbf{W}$. For $\mathbf{W}'\mathbf{X}'\mathbf{V}$ the space cost is $O(nd)$. Put everything together the space cost for computing $\nabla_{\mathbf{U}}\mathcal{L}$ is $O(m + nd)$.

The time complexity of the part within $[\cdot]$ in Eq.(3.16) is $O(nd^2+md)$. The time complexity of computing $\mathbf{W}'\mathbf{X}'\mathbf{V}$ is $O(ndr)$. The time complexity about the multiplication between $[\cdot]$ and $\mathbf{W}'\mathbf{X}'\mathbf{V}$ is $O(ndr)$. Hence, putting everything together, the total time complexity for computing $\nabla_{\mathbf{U}}\mathcal{L}$ is $O(nd^2 + md)$ given $r \ll d$. QED.

3.1.4 Experiments

Setups. We evaluate the proposed GaSoliNe on Cora, Citeseer, and Polbolgs datasets [20, 104, 114]. Since the Polblogs dataset does not contain node features, an identity matrix is used as the node feature matrix. All the datasets are undirected unweighted graphs and we experiment on the largest connected component of every dataset.

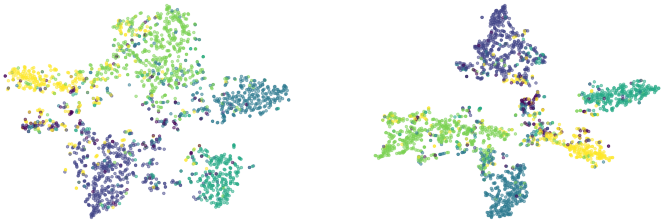
To set fair modification budgets across different datasets, the modification budget on adjacency matrix $B_{\texttt{topo}}$ is defined as $B_{\texttt{topo}} = m \times \texttt{modification rate}_{\texttt{topo}}$ and the budget on feature matrix $B_{\texttt{fea}}$ is defined as $B_{\texttt{fea}} = n \times d \times \texttt{modification rate}_{\texttt{fea}}$.
where m, n, and d are the number of edges, nodes, and node features. In all the experiments, $\texttt{modification rate}_{\texttt{topo}} = 0.1$ and $\texttt{modification rate}_{\texttt{fea}} = 0.001$. Detailed hyperparameter settings are studied in the following content. We report the mean $\pm$ std accuracy over 10 repetitions as the evaluation metric.

Applicability of GaSoliNe. The proposed GaSoliNe trains a *backbone classifier* in the lower-level problem and uses the trained backbone classifier to modify the initially provided graph and improve the performance of the *downstream classifier* on the test nodes. In addition, GaSoliNe is capable of modifying both the graph topology (i.e., $\mathbf{A}$) and feature (i.e., $\mathbf{X}$) in both the discretized and continuous fashion. To verify that, we select three classic GNNs-based node classifiers, including GCN [20], SGC [22], and APPNP [23] as the backbone classifiers and the downstream classifiers. The detailed experiment procedure is as follows. First, we modify the given graph using the proposed GaSoliNe algorithm with 4 modification strategies (i.e., modifying topology or node feature with discretized or continuous modification). Each variant is implemented with 3 backbone classifiers so that in total, there are 12 sets of GaSoliNe settings. Second, for every of the 12 modified graphs, we evaluate 3 downstream classifiers and report the result (mean$\pm$std accuracy). For this subsection, the initially provided graph is Citeseer [20].

Experimental results are reported in Table 3.3 where 'DT', 'CT', 'DF', and 'CF' denote 'discretized topology', 'continuous topology', 'discretized feature', and 'continuous feature' modifications, respectively. The second row of Table 3.3 shows the results on the initially provided graph, and the other rows denote the results on modified graphs with different

Table 3.3: Effectiveness of GASOLINE under multiple variants (mean±std accuracy). The first and second columns denote the modification strategies and backbone classifiers adopted by GASOLINE, respectively. The remaining columns show the performance of various downstream classifiers. • indicates significant improvement compared with results on the original graph (values at the second row) with a p-value¡0.01 ,and ∘ indicates no statistically significant improvement.

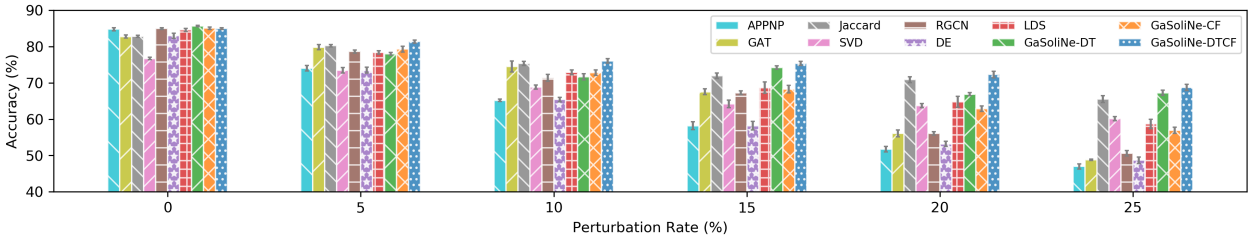
Variant	**Backbone**	**GCN**	**SGC**	**APPNP**
None	None	72.2±0.5	72.8±0.2	71.8±0.4
DT	GCN	74.7±0.3•	74.8±0.1•	75.4±0.2•
	SGC	74.7±0.4•	75.2±0.2•	75.6±0.3•
	APPNP	74.6±0.3•	74.6±0.1•	75.4±0.4•
DF	GCN	72.4±0.3∘	72.7±0.2∘	72.8±0.4•
	SGC	73.3±0.5•	73.4±0.2•	73.6±0.4•
	APPNP	72.6±0.3∘	72.9±0.1∘	73.6±0.4•
CT	GCN	73.1±0.4•	73.6±0.1•	74.8±0.2•
	SGC	73.0±0.3•	73.5±0.2•	74.4±0.3•
	APPNP	72.8±0.5∘	73.4±0.1•	74.4±0.9•
CF	GCN	72.7±0.4∘	73.6±0.1•	73.8±0.3•
	SGC	72.9±0.4•	73.6±0.4•	73.8±0.4•
	APPNP	73.0±0.3•	73.6±0.2•	73.9±0.3•



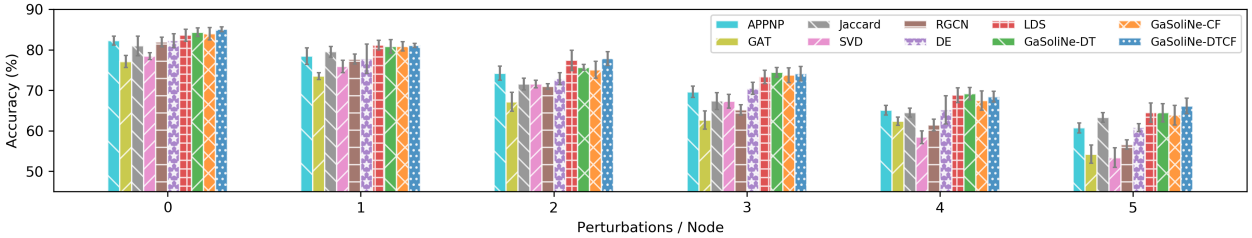
(a) Original (b) After modification

Figure 3.1: Visualization of node embeddings from original Citeseer graph (a) and modified Citeseer graph by GASOLINE (b). It is best viewed in color.

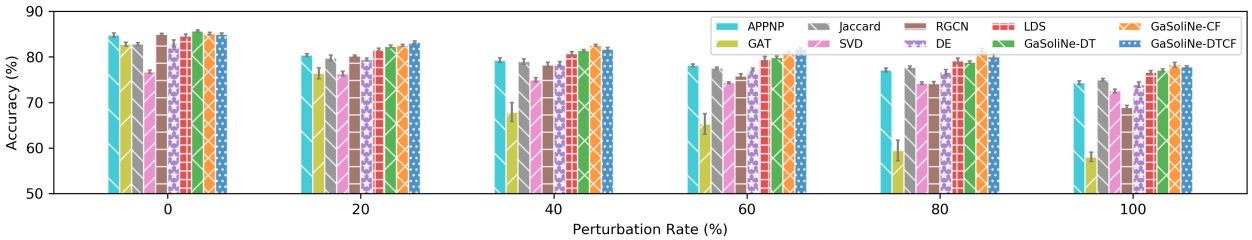
settings. We use • to indicate that the improvement of the result is statistically significant compared with results on the initially provided graph with a p-value$<$ 0.01, and we use ∘ to indicate no statistically significant improvement. We have the following observations. First, in the vast majority cases, the proposed GASOLINE is able to statistically significantly improve the accuracy of the downstream classifier over the initially-provide graph, for every combination of the modification strategy (discretized vs. continuous) and the modification target (topology vs. feature). Second, the graphs modified by GASOLINE with different backbone classifier can benefit different downstream classifiers, which demonstrates great transferability and broad applicability.



(a) metattack



(b) NETTACK



(c) random attack

Figure 3.2: Performance comparison between baselines and GASOLINE under (a) metattack, (b) NETTACK, and (c) random attack with various perturbation rates. It is best viewed in color.

We further provide visualization of node embeddings before and after modification. We present the visualizations of the initial Citeseer graph and the modified Citeseer graph from GASOLINE DT variant with SGC [22] backbone classifier. The detailed procedure is that a GCN [20] is trained on the training set $\mathcal{T}$ of given initial/modified graphs, and hidden representations are used as node embeddings. Then, t-SNE [118] maps the node embeddings into two-dimensional ones for visualization. Figure 3.1 shows the visualization results from the original and the modified Citeseer graphs. Clearly, the node embeddings from the modified graph are more discriminative than the embeddings from the original graph. Specifically, the clusters are more cohesive and there is less overlap between clusters in the modified graphs (i.e., Figures 3.1b) compared with those on the original graph (i.e., Figure 3.1a). It further demonstrates that even though we do not know the downstream classifiers (in this case, the backbone classifier and downstream classifier are different), the proposed GASOLINE can still improve the graph quality to benefit downstream classifiers.

Table 3.4: Comparison with baselines on heavily poisoned datasets (mean±std accuracy). Some results are not applicable since Jaccard requires node features that are absent on the Polblogs graph. G denotes GaSoliNe for short.

Attack	Data	APPNP	GAT	Jaccard	SVD	RGCN	DE	LDS	G-DT	G-CF	G-DTCF
metattack	Cora	47.0±0.7	48.8±0.2	65.4±0.9	60.3±0.8	50.6±0.8	48.7±0.9	58.7±1.3	67.3±0.7	57.0±0.9	**68.8±0.9**
	Citeseer	49.4±2.2	62.4±0.7	57.1±1.0	49.5±0.8	55.5±1.4	50.1±2.3	58.2±2.3	**63.5±1.5**	58.4±1.5	62.2±1.0
	Polblogs	58.4±3.6	48.2±6.6	N/A	**79.1±2.4**	50.8±0.9	56.4±6.3	63.7±5.7	65.0±0.7	55.0±4.1	64.7±1.4
Nettack	Cora	60.7±1.2	54.2±2.3	63.7±1.4	52.9±2.8	56.5±1.1	60.8±1.0	64.5±2.4	64.5±2.2	63.9±2.4	**66.1±1.9**
	Citeseer	68.3±6.8	61.9±4.4	72.5±3.3	50.2±6.6	56.4±1.5	63.3±4.7	71.0±3.3	71.6±3.9	69.4±4.8	**74.3±1.6**
	Polblogs	90.5±1.0	91.1±0.7	N/A	**93.6±1.2**	93.1±0.2	89.1±2.4	91.1±1.8	92.3±1.6	90.3±0.7	92.4±1.7
random attack	Cora	74.3±0.4	58.1±1.0	75.1±0.5	72.6±0.3	68.9±0.4	73.9±0.6	76.6±0.4	77.1±0.3	**78.3±0.5**	77.8±0.2
	Citeseer	69.8±0.6	60.8±1.6	69.7±0.5	66.7±0.4	65.7±0.2	69.4±0.5	72.3±0.4	**73.8±0.2**	72.3±0.4	73.4±0.5
	Polblogs	74.7±2.8	**84.5±1.0**	N/A	83.3±2.8	81.7±0.9	75.9±1.4	73.2±2.8	73.4±4.1	77.1±1.6	77.6±2.9

Effectiveness of GaSoliNe. The defects of the initially-provided graph can be due to various reasons, such as construction bias or even malicious poisoning. In this subsection, we evaluate the effectiveness of the proposed GaSoliNe by (A) comparing baseline methods on various poisoned/noisy graphs and (B) integrating with existing robust GNNs methods. The attack methods we adopt are as follows: (1) Random Attack randomly flips entries of benign adjacency matrices with different `perturbation rate`; (2) Nettack [114] attacks a set of target nodes with different `perturbations/node`; (3) metattack [104] poisons the performance of node classifiers by perturbing the overall benign graph topology with different `perturbation rate`.

A - Comparison with Baseline Methods. We compare GaSoliNe with the following baseline methods: APPNP [23], GAT [21], Jaccard [101], SVD [100], RGCN [103], DE [119], and LDS [54]. Recall that we feed all the graph modification-based methods (Jaccard, SVD, DE, LDS, GaSoliNe) with the exactly same downstream classifier (APPNP) for a fair comparison.

We set 3 variants of GaSoliNe to compare with the above baselines. To be specific, we refer to (1) GaSoliNe with discretized modification on topology as GaSoliNe-DT, (2) GaSoliNe with continuous modification on feature as GaSoliNe-CF, and (3) GaSoliNe with discretized modification on topology and continuous modification on feature as GaSoliNe-DTCF. All these GaSoliNe variants use APPNP [23] as both the backbone classifier and the downstream classifier. We test various perturbation rates (i.e., `perturbation rate` of metattack from 5% to 25% with a step of 5%, `perturbation rate` of random attack from 20% to 100% with a step of 20%, and `perturbations/node` of Nettack from 1 to 5) to attack the Cora [20] dataset and report the accuracy (mean±std) in Figure 3.2. From experiment results we observe that: (1) with the increase of adversarial perturbation, the performance of all methods drops, which is consistent with our intuition; (2) variants of

GaSoliNe consistently outperform the baselines under various adversarial/noisy scenarios; and (3) the proposed GaSoliNe even improves over the original, benign graphs (i.e., 0 `perturbation rate` and 0 `perturbations/node`).

An interesting question is, if the initially-provided graph is heavily poisoned/noisy, to what extent is the proposed GaSoliNe still effective? To answer this question, we study the performance of GaSoliNe and other baseline methods on heavily-poisoned graphs (100% `perturbation rate` of random attack, 25% `perturbation rate` of metattack, and 5 `perturbations/node` of Nettack). The detailed experiment results are presented in Table 3.4. In most cases, GaSoliNe can obtain competitive or even better performance against baseline methods. On the Polblogs graph, GaSoliNe does not perform as well as in the other two datasets. This is because, (1) the Polblogs graph does not have node feature which weakens the effectiveness of modification from GaSoliNe and (2) the Polblogs graph has strong low-rank structure, which can be further verified in the following experiments. As flexible solutions, in the following subsection, we study whether GaSoliNe can work together with other graph defense methods.

B - Incorporating with Graph Defense Strategies. GaSoliNe does not make any assumption about the properties of the defects of the initially-provided graph. We further evaluate if GaSoliNe can boost the performance of both model-based and data-based defense strategies under the heavily-poisoned settings. We use a data-based baseline SVD [100], a model-based baseline RGCN [103], and another strong baseline GAT [21] to integrate with GaSoliNe since they have shown competitive performance from Table 3.4 and Figure 3.2. The detailed procedure for model-based methods (GAT and RGCN) is as follows: GaSoliNe first modifies the graph, and then the baselines are implemented on the modified graph to report the final results. For the data-based method (SVD), the baseline is implemented to preprocess graphs first, and then we modify graphs again by GaSoliNe, and finally run the downstream classifiers (APPNP) on the twice-modified graphs. In this task, GaSoliNe-DTCF is adopted. In order to heavily poison the graphs, we use metattack [104] with `perturbation rate` as 25% to attack the benign graphs. We report the results in Table 3.5 and observe that after integrating with GaSoliNe, performance of all the defense methods further improves significantly with a p-value¡0.01.

Efficacy of Low-Rank GaSoliNe. To answer RQ3, we first compare the performance of APPNP [23] on two modified graphs from the low-rank GaSoliNe (denoted as GaSoliNe-LR) and the original GaSoliNe, respectively. Specifically, we adopt its variant for the original GaSoliNe with continuous modification towards the network topology. Due to the space limitation, we only show the results given graphs perturbed by metattack [104]

Table 3.5: Node classification accuracy (mean±std) boosting of existing defense methods on poisoned graphs (25% edges perturbed by metattack [104]) by the proposed GaSoliNe.

Data	With GaSoliNe?	GAT	SVD	RGCN
Cora	N	48.8±0.2	60.3±0.8	50.6±0.8
	Y	63.7±0.6	79.7±0.6	62.6±0.6
Citeseer	N	62.4±0.7	49.5±0.8	55.5±1.4
	Y	69.7±0.2	76.5±0.6	66.1±0.8
Polblogs	N	48.2±6.6	79.1±2.4	50.8±0.9
	Y	70.8±0.6	89.2±0.7	67.7±0.3

in Table 3.6. We observe that in most settings on both the Cora and Citeseer datasets, the GaSoliNe-LR can obtain promising performance against the original GaSoliNe. Surprisingly, on the Polblogs dataset, the GaSoliNe-LR shows great advantages over the original GaSoliNe. One possible explanation is that the Polblogs dataset is inherently low-rank (which can be corroborated by Table 3.4 where SVD [100] obtains strong performance) and GaSoliNe-LR learns a low-rank incremental matrix which amplifies the advantage further.

To verify the efficiency of the proposed GaSoliNe-LR, we generate a set of synthetic graphs with different numbers of nodes n. The wall-clock time for computing the hypergradient is presented in Figure 3.3. Clearly, the GaSoliNe-LR is much more efficient than the original GaSoliNe especially when the network size is large.

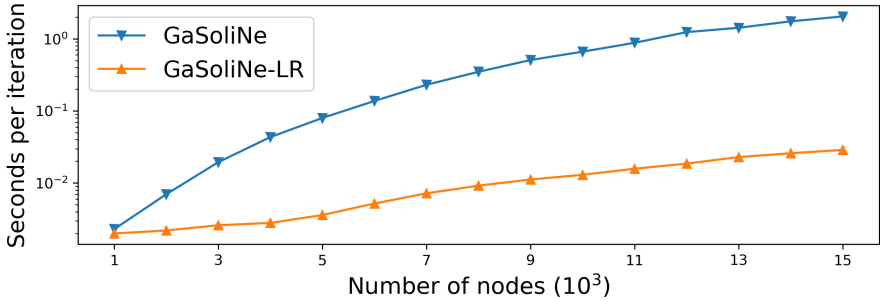


Figure 3.3: Efficiency comparison between GaSoliNe and GaSoliNe-LR

Case Study about the Behavior of GaSoliNe. Here, we further study the potential reasons behind the success of GaSoliNe. To this end, we conduct a case study whose core idea is to label malicious modifications (from adversaries) and test if GaSoliNe can detect them. The specific procedure is that we utilize different kinds of attackers (i.e., metattack [104], Nettack [114], and random attack) to modify the graph structure of a *benign* graph G (with adjacency matrix $\mathbf{A}$) into a *poisoned* graph $G_{\texttt{adv}}$ (with adjacency

Table 3.6: Effectiveness comparison between GaSoliNe and GaSoliNe-LR

Data	Ptb Rate (%)	APPNP	GaSoliNe	GaSoliNe-LR
Cora	0	84.0±0.4	85.2±0.2	84.4±0.3
	5	74.1±0.7	77.4±0.5	75.0±0.3
	10	65.2±0.4	70.8±0.5	67.9±0.9
	15	58.2±1.1	67.1±0.8	65.3±0.8
	20	51.7±0.7	62.5±0.5	60.2±1.2
	25	47.0±0.7	57.3±0.6	57.1±0.5
Citeseer	0	71.8±0.4	74.7±0.2	73.4±0.2
	5	67.6±0.9	69.6±0.7	68.2±0.8
	10	61.8±0.8	66.3±1.0	63.9±0.4
	15	54.1±0.8	59.3±1.1	56.8±1.1
	20	51.0±1.2	56.5±0.9	55.3±0.9
	25	49.4±2.2	57.7±1.8	56.5±0.8
Polblogs	0	94.1±0.6	95.3±0.6	95.7±0.3
	5	70.1±0.6	73.8±0.9	93.4±0.3
	10	69.8±0.8	72.8±0.4	90.8±0.2
	15	67.5±0.5	70.1±1.2	88.7±0.3
	20	64.1±0.9	68.5±1.0	88.0±0.3
	25	57.0±3.6	64.8±2.1	89.9±0.5

matrix $\mathbf{A}_{\texttt{adv}}$). Then, we utilize the score matrix $\mathbf{S}$ from Eq. (3.13) to assign a score to every entry of the poisoned adjacency matrix $\mathbf{A}_{\texttt{adv}}$. As we mentioned in Section 3.1.3, the higher score an entry obtains, the more likely GaSoliNe will modify it. We compute the average score of three groups of entries from $\mathbf{A}_{\texttt{adv}}$: the poisoned entries after adding/deleting perturbations from adversaries, the benign existing edges without perturbation, and the benign non-existing edges without perturbation. Note that both the benign graphs and the poisoned graphs are unweighted, and we define the following auxiliary matrices. $\mathbf{A}_{\texttt{diff}} = |\mathbf{A}_{\texttt{adv}} - \mathbf{A}|$ is a difference matrix whose entries with value 1 indicate poisoned entries. $\mathbf{A}_{\texttt{benign-E}} = \mathbf{A} \odot (\mathbf{1} - \mathbf{A}_{\texttt{diff}})$ is a benign edge indicator matrix whose entries with value 1 indicate the benign existing edges without perturbation. $\odot$ indicates element-wise multiplication. $\mathbf{A}_{\texttt{benign-NE}} = (\mathbf{1}-\mathbf{A}) \odot (\mathbf{1}-\mathbf{A}_{\texttt{diff}})$ is a benign non-existing edge indicator matrix whose entries with value 1 indicate the benign non-existing edges without perturbation.

Based on that, we have the following three statistics:

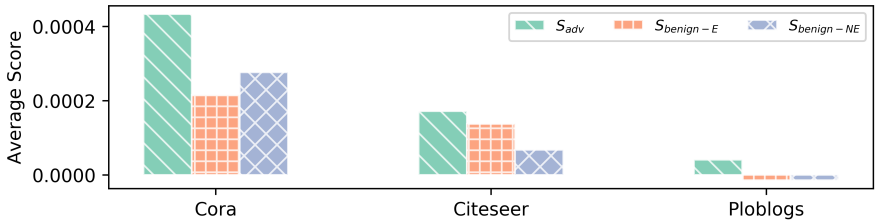
$$S_{\texttt{adv}} = \frac{\sum_{i,j} \left(\mathbf{S} \odot \mathbf{A}_{\texttt{diff}}\right)[i,j]}{\sum_{i,j} \mathbf{A}_{\texttt{diff}}[i,j]}, \tag{3.17}$$

$$S_{\texttt{benign-E}} = \frac{\sum_{i,j} \left(\mathbf{S} \odot \mathbf{A}_{\texttt{benign-E}}\right)[i,j]}{\sum_{i,j} \mathbf{A}_{\texttt{benign-E}}[i,j]}, \tag{3.18}$$

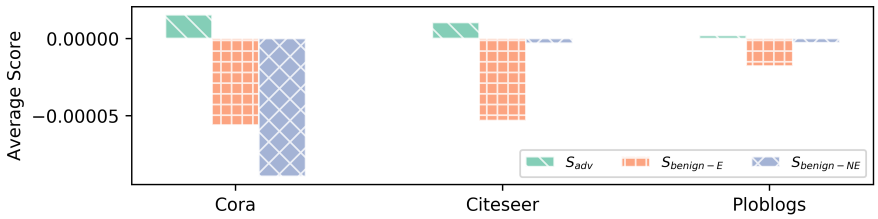
$$S_{\texttt{benign-NE}} = \frac{\sum_{i,j} \left(\mathbf{S} \odot \mathbf{A}_{\texttt{benign-NE}}\right)[i,j]}{\sum_{i,j} \mathbf{A}_{\texttt{benign-NE}}[i,j]}, \tag{3.19}$$

which denotes the average score obtained by poisoned entries, benign existing edges, and benign non-existing edges.

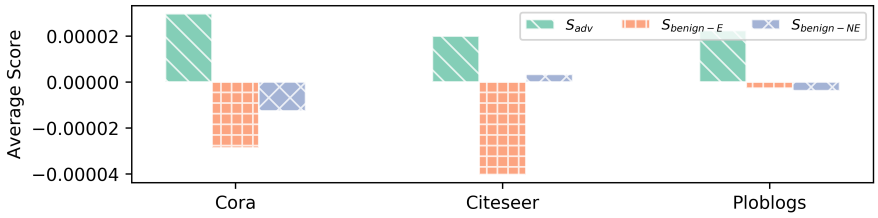
Detailed results are presented in Figure 3.4. We observe that GASOLINE tends to modify poisoned entries more (with higher scores) than to modify benign unperturbed entries in the adjacency matrix of poisoned graphs, which is consistent with our expectation and enables the algorithm to recover the benign graphs partially and to boost the performance of downstream classifiers.



(a) metattack



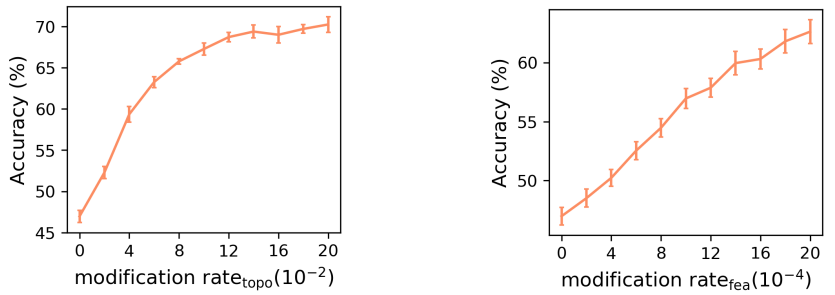
(b) NETTACK



(c) random attack

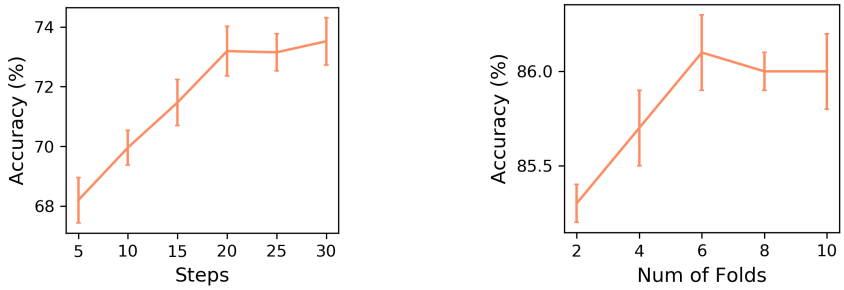
Figure 3.4: Score of various entries under metattack (a), NETTACK (b), and random attack (c). Best viewed in color.

Effect of Modification Budget. In this section, we study the relationships between the budget of GASOLINE and the corresponding performance of the downstream classifier. Here, we instantiate two variants of GASOLINE: discretized modification towards topology (GASOLINE-DT) and continuous modification towards feature (GASOLINE-CF). The provided graph is Cora [20] which is heavily-poisoned by metattack [104] with `perturbation rate` = 25% (i.e., B). The perturbation budget per modification step b is set to be $\frac{B}{10}$. Both the backbone classifier and the downstream classifier of GASOLINE are the APPNP [23] models with the settings above. From Figure 3.5 we observe that with the increase of the budget (`modification rate`$_{\texttt{topo}}$ and `modification rate`$_{\texttt{fea}}$), GASOLINE enjoys great potential to improve the performance of the downstream classifiers further. At the same time, 'economic' choices are strong enough to benefit downstream classifiers, so we set `modification rate`$_{\texttt{topo}}$ as 0.1 and `modification rate`$_{\texttt{fea}}$ as 0.001 throughout our experiment settings.



(a) GASOLINE-DT (b) GASOLINE-CF

Figure 3.5: Performance of downstream classifier vs. the modification budget of GASOLINE-DT (a) and GASOLINE-CF (b)



(a) update steps (b) number of folds K

Figure 3.6: Performance of GASOLINE-DTCF vs. the update steps (a) and the number of folds K (b).

Effects of Modification Steps and Number of Folds. As stated in the main content, during implementation, we set the budget to b in every iteration and update the given

graph multiple times until the total budget B is exhausted. Hence, the update steps are equal to $\lceil \frac{B}{b} \rceil$. Intuitively, a smaller budget per iteration can provide a finer update towards the given graphs. To validate that we test the performance of an instantiation of GaSoliNe with discretized modification towards topology and continuous modification towards feature (GaSoliNe-DTCF) on the Cora [20] graph which is poisoned by metattack [104] with `perturbation rate` = 25%. Both the backbone classifier and the downstream classifier of GaSoliNe are the APPNP [23] with the settings above. From Figure 3.6a, we observe that with more update steps, downstream classifiers can get better performance. However, when the number of steps exceeds 20, the performance improvement is minor.

Additionally, the number of training/validation split folds, K, is another important hyperparameter in our model. Intuitively, a larger K leads to better usage of the given data. To study the relationships between K and the corresponding performance of the downstream classifier, we implement GaSoliNe-DTCF on the original Cora graph to verify that. Note that the $\texttt{modification rate}_{\texttt{topo}} = 0.1$, $\texttt{modification rate}_{\texttt{fea}} = 0.001$, and the number of modification steps is set as 10. From Figure 3.6b we observe that the performance of the downstream classifier is improved with the increase of the number of folds. However, such performance gains stop when $K = 6$. Hence, $K = 6$ is enough to make full use of the given graph by GaSoliNe.

3.2 GENERALIZED FEW-SHOT NODE CLASSIFICATION

3.2.1 Introduction

The task of node classification aims to classify nodes into categories, which has been extensively studied [20, 21, 23, 31, 120, 121, 122, 123]. Typically, sufficient labeled nodes from *all* the classes are the cornerstone of this task. However, due to the ever-growing new data and high annotation cost by human labor, the number of labeled nodes from various classes tends to follow a long-tailed distribution [124]. Consequentially, some classes might not have sufficient labeled nodes, which in turn degrades the performance of node classifiers dramatically. This problem with limited labeled nodes per class (i.e., shots) is known as few-shot node classification [124, 125, 126, 127, 128].

Formally, few-shot node classification separates the classes of interest into the base and novel classes, where the former are provided with many shots and the latter are provided with only few shots (usually less than 10). At the test phase, the classifier aims to accurately classify test nodes *only* into the novel classes. Directly training advanced node classifiers (e.g., graph neural networks (GNNs)) on few labeled nodes from novel classes is prone to

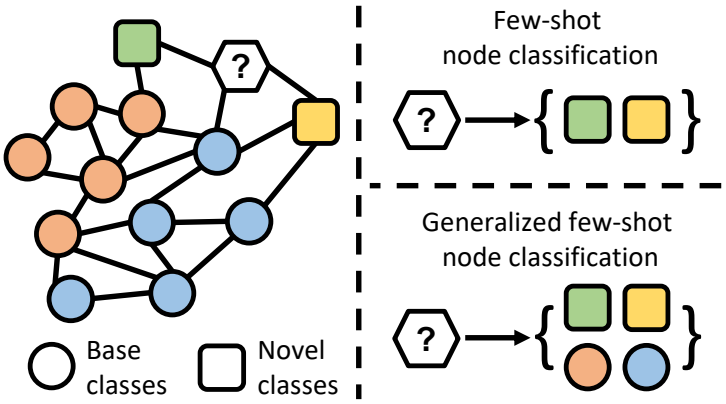


Figure 3.7: An illustrative example to show the difference between the few-shot and generalized few-shot node classification tasks. For the generalized one, the test node is expected to be classified into the joint set of the novel and base classes.

overfitting. Recently, increasing research efforts have been made to address the few-shot node classification problem. Representative works include MetaGNN [125], graph prototypical networks (GPN) [124], G-META [127], and so on. Most of the existing solutions are under the umbrella of *meta-learning* [129], specifically, the strategy named episodic training [124, 125, 130, 131, 132]. Concretely, they first construct episodes from the base classes, every of which includes the training and evaluation of a (base-) learner (e.g., a node classifier). Then, a meta-learner is optimized by supervising the training and evaluation process of the (base-) learner, and 'learns to X' (e.g., learns to train a node classifier) through the iterative episodes. With such high-level knowledge extracted by the episodic training, the meta-learner can effectively assist the node classifier to handle the node classification tasks on few-shot novel classes.

Despite the great progress, most, if not all, of them follow a strong assumption that all the test nodes must be exclusively sampled from the novel classes, which is hardly realistic in real-world applications. A motivating example is a bibliography literature classification system, whose nodes represent published literature and links represent reference relationships. More often than not, a newly-published paper could fall into a classic domain (i.e., a base class) instead of a new domain (i.e., a novel class). To bridge the gap between the setting of the few-shot node classification problem and real-world scenarios, this paper studies a broader and more practical problem, namely *generalized few-shot node classification*. Given base classes with many shots and novel classes with few shots, in this new problem, a node classifier is expected to perform classification on the *joint label set* of both base and novel classes, instead of the label set of the novel classes alone. An illustration is provided in Fig. 3.7.

This subtle difference in the target label set leads to a significantly more challenging problem from two key perspectives.

Challenges. First (*asymmetric classification*), at the meta-test phase for meta-learning-based methods (or the test phase for standard learning methods), a classifier tends to show more confidence towards the base classes compared with the novel classes [133] due to the imbalanced shots [134, 135]. In Table 3.7, we illustrate that by presenting classification results on Amazon-Clothing [136] and Cora-Full [137] datasets from a classic node classifier (APPNP [23]) and a few-shot node classifier (MetaGNN [125]). We observe that the majority of nodes from the novel classes are classified into the base classes, which will definitely lead to misclassification. Similar results have been reported in the generalized zero-shot learning problem [138, 139, 140] from the computer vision domain.

Second (*inconsistent preference*), appropriately aggregating information from multiple receptive fields is the key operation for the effectiveness of graph neural networks [23, 121, 141]. However, the optimal weight assignments among receptive fields could vary dramatically between the many-shot cases (i.e., from a base class) and the few-shot cases (i.e., from a novel class). For instance, on a sparsely-connected homophilic graph (where edges often connect same-class nodes), under the many-shot settings, classifiers prefer to pay more attention to the local information [23, 121] from small receptive fields; on the contrary, under the few-shot settings, more attention to the long-range propagation is necessary as the labeled nodes will locate sparsely in the given graph [23, 70, 126, 142]. There is tension between the above two scenarios about the weight assignments of receptive fields. Nonetheless, it remains unknown how to design an adaptive model that can kill two birds with one stone, addressing both base and novel classes. It is worth noting that the inconsistent preference challenge only applies to the generalized few-shot node classification problem. In contrast, for the standard few-shot node classification problem, a classifier will not face the *inconsistent preference* challenge, and a consistent weight assignment is often sufficient. This is because, at the meta-test time, classifiers only need to predict novel classes whose numbers of shots are more or less balanced.

3.2.2 Preliminaries

This section introduces the notations, the formal definition of the problem we aim to study, and preliminary works on few-shot learning based on meta-learning.

Notations. We use bold uppercase letters for matrices (e.g., $\mathbf{A}$), bold lowercase letters for vectors (e.g., $\mathbf{u}$), lowercase and uppercase letters in the regular font for scalars (e.g., d, K),

Table 3.7: Test nodes (%) classified from base classes to base classes (i.e., $b \to b$), from base classes to novel classes (i.e., $b \to n$), from novel classes to base classes (i.e., $n \to b$), and from novel classes to novel classes (i.e., $n \to n$), respectively.

Dataset	**Method**	$b \to b$	$b \to n$	$n \to b$	$n \to n$
Amazon Clothing	APPNP	100.0	0.0	69.6	30.4
	MetaGNN	100.0	0.0	61.2	38.8
Cora-Full	APPNP	99.2	0.8	66.8	33.2
	MetaGNN	99.0	1.0	72.4	27.6

and calligraphic letters for sets (e.g., $\mathcal{T}$). $\mathbf{A}[i,j]$ denotes the entry of matrix $\mathbf{A}$ at the i-th row and the j-th column, $\mathbf{A}[i,:]$ denotes the i-th row of matrix $\mathbf{A}$, and $\mathbf{A}[:,j]$ denotes the j-th column of matrix $\mathbf{A}$. Similarly, $\mathbf{u}[i]$ denotes the i-th entry of vector $\mathbf{u}$. Superscript $\top$ denotes the transpose of matrices and vectors (e.g., $\mathbf{A}^\top$ is the transpose of $\mathbf{A}$). An attributed graph can be represented as $\mathcal{G} = \{\mathbf{A}, \mathbf{X}\}$ which is composed by an adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ and an attribute matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, where n is the number of nodes and d is the node feature dimension. In total, nodes can be categorized into a set of classes $\mathcal{C}$. The node set $\mathcal{V}$ and the class set $\mathcal{C}$ will be split and notated with appropriate subscripts. For example, $\mathcal{V}_{\texttt{test}}$ and $\mathcal{C}_{\texttt{novel}}$ refer to the test nodes and the novel classes, respectively. N denotes the number of novel classes and K denotes the number of training nodes per novel class. Note that the number of base classes and the number of training nodes per base class are much larger than N and K, and they are not fixed across different datasets.

Problem Definition. As mentioned in Section 3.2.1, we do not exclude the base classes from the class membership of test nodes and study the *generalized few-shot node classification* problem which is defined as follows.

Problem 3.2. Generalized few-shot node classification
Given: (1) a graph $\mathcal{G} = \{\mathbf{A}, \mathbf{X}\}$, (2) labeled nodes $\mathcal{V}_{\texttt{base}}$ from base classes $\mathcal{C}_{\texttt{base}}$, (3) labeled nodes $\mathcal{V}_{\texttt{novel}}$ from novel classes $\mathcal{C}_{\texttt{novel}}$ ($\mathcal{C}_{\texttt{base}} \cap \mathcal{C}_{\texttt{novel}} = \emptyset$) where each novel class has very few (e.g., 1) labeled nodes, (4) unlabelled test nodes $\mathcal{V}_{\texttt{test}}$ from classes $\mathcal{C}_{\texttt{novel}} \cup \mathcal{C}_{\texttt{base}}$, where $\mathcal{V}_{\texttt{base}} \cap \mathcal{V}_{\texttt{test}} = \emptyset$ and $\mathcal{V}_{\texttt{novel}} \cap \mathcal{V}_{\texttt{test}} = \emptyset$.
Find: The predicted labels for the unlabelled test nodes $\mathcal{V}_{\texttt{test}}$.

We remark that the topology and attribute information of all the nodes are given and we study this problem under the semi-supervised (and transductive) setting.

Based on the naming convention from the few-shot learning community, we name it as the *generalized N-way K-shot* node classification problem. Note that this naming convention

only describes the setting of novel classes. The number of base classes is at least $3N$, and each of the base classes is provided with at least $10K$ shots.

Episodic Training for Meta-Learning. Meta-learning-based few-shot learning solutions inspire our proposed method. Here, we introduce the classic episodic training [124, 125, 130, 131, 132, 143].

Meta-learning is also known as *learning-to-learn* which describes the interaction between a meta-learner (parameterized by ϕ) and a (base-) learner (e.g., a classifier parameterized by θ). According to the conventional N-way K-shot problem setting [124, 125, 132], at the *meta-test* phase, all the nodes exclusively come from the novel classes $\mathcal{C}_{\texttt{novel}}$ and $|\mathcal{C}_{\texttt{novel}}| = N$. The classifier θ will fit on the provided $N \times K$ labeled nodes (i.e., K labeled nodes per novel class) with the assistance of the meta-learner ϕ. The meta-test performance is measured by the fitted classier on the test nodes.

To align the meta-training and meta-test scenarios, episodic training [124, 125, 130, 131, 132] mimics the meta-test scenario and generates episodes $\{\mathcal{E}_i = \{\mathcal{S}_i, \mathcal{Q}_i\}\}$ from base classes. In every episode, N base classes are randomly selected. Then, for the selected base classes, K and I labeled nodes per base class are sampled to compose the support set $\mathcal{S}_i$ and the query set $\mathcal{Q}_i$ respectively. Here the configuration of the support set is to align with the N-way K-shot meta-test scenarios, and I is fixed (e.g., 30) as many existing works [124, 131] did. As the support set $\mathcal{S}_i$ and query set $\mathcal{Q}_i$ are both labeled but do not overlap with each other, we use v and v' with indices to distinguish nodes from the support set $\mathcal{S}_i$ with those from the query set $\mathcal{Q}_i$. The i-th episode can be represented as Eq. (3.20) and the training objective can be formulated as Eq. (3.21).

$$\mathcal{S}_i = \{v_1, \ldots, v_{N\times K}\}, \ \mathcal{Q}_i = \left\{v'_1, \ldots, v'_{N\times I}\right\}. \tag{3.20}$$

$$\begin{aligned} \phi^* &= \arg\min_{\phi} \ \mathbb{E}_{v_i \in \mathcal{Q}} \ L_{\texttt{cla}} \left(z\left(\mathcal{G}, \theta^*, \phi, v_i\right), y_i\right), \\ s.t.\ \theta^* &= \arg\min_{\theta} \mathbb{E}_{v_j \in \mathcal{S}} \ L_{\texttt{cla}} \left(z\left(\mathcal{G}, \theta, \phi, v_j\right), y_j\right), \end{aligned} \tag{3.21}$$

where $z(\mathcal{G}, \theta, \phi, v_i)$ is the classification results on node v_i by the classifier θ (with the assistance from the meta-learner ϕ), y_i is the label of v_i, and $L_{\texttt{cla}}()$ is the classification loss. The classifier θ is trained from scratch with the meta-learner ϕ on the support set $\mathcal{S}_i$ (i.e., the lower-level objective) and its loss on the query set $\mathcal{Q}_i$ serves as the supervision to update the meta-learner ϕ (i.e., the upper-level objective).

3.2.3 Proposed Method

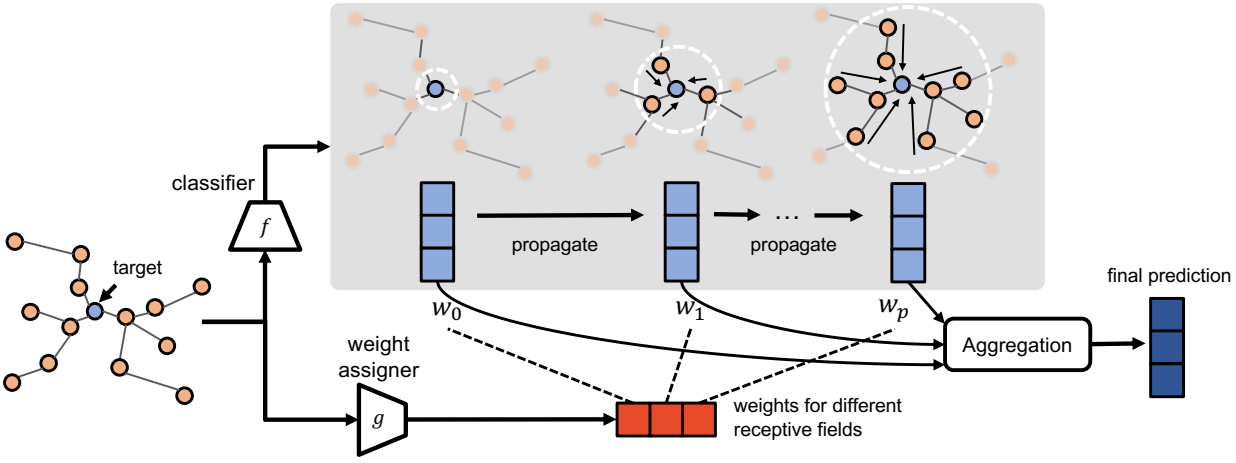


Figure 3.8: The framework of STAGER. The propagation step $p = 2$ in this figure.

In this section, we first introduce the overall motivation of our model design. Then, we present the concrete instantiation of every model component. After that, a tailored, imbalanced episodic training approach is proposed for our model.

Design Motivation. From the statistical learning perspective, the goal of the generalized few-shot node classification is to infer the probability $P(y_i|\mathcal{G}, v_i)$, where y_i is the label of the node v_i:

$$P\left(y_i|\mathcal{G}, v_i\right) = \sum_{\tilde{C}_i \in \{novel, base\}} P\left(y_i|\tilde{C}_i, \mathcal{G}, v_i\right) P\left(\tilde{C}_i|\mathcal{G}, v_i\right), \tag{3.22}$$

where $\tilde{C}_i$ is a variable to indicate whether the given node v_i belongs to novel classes or base classes. When $\tilde{C}_i = base$, inferring $P\left(y_i|\tilde{C}_i, \mathcal{G}, v_i\right)$ is equivalent to the standard node classification problem [20, 21, 23, 120, 121, 122, 123] with many shots. On the contrary, if $\tilde{C}_i = novel$, inferring $P\left(y_i|\tilde{C}_i, \mathcal{G}, v_i\right)$ is equivalent to the few-shot node classification problem [124, 125, 132]. In a nutshell, there exist rich approaches to estimate $P\left(y_i|\tilde{C}_i, \mathcal{G}, v_i\right)$ in both cases.

However, resolving the problem in one model as Eq. (3.22) is a great challenge. The empirical evidence in Table 3.7 illustrates that the effect of *asymmetric classification* deteriorates the classification performance significantly. In other words, when applied to the generalized few-shot node classification problem, existing classic and few-shot node classifiers tend to over-estimate $P\left(\tilde{C}_i = base|\mathcal{G}, v_i\right)$ yet under-estimate $P\left(\tilde{C}_i = novel|\mathcal{G}, v_i\right)$, i.e., $P\left(\tilde{C}_i = base|\mathcal{G}, v_i\right) \gg P\left(\tilde{C}_i = novel|\mathcal{G}, v_i\right)$ even if v_i is from the novel classes. An intuitive

explanation is that, for existing methods, most of them directly estimate $P(y_i|\mathcal{G}, v_i)$ by one parametric model whose training is overwhelmed by the labeled nodes from the base classes. Thus, the implicit component $P\left(\tilde{C}_i|\mathcal{G}, v_i\right)$ of the estimated probability $P(y_i|\mathcal{G}, v_i)$ is heavily biased. Based on the above analysis, our overall solution for the *asymmetric classification* is estimating $P\left(y_i|\tilde{C}_i, \mathcal{G}, v_i\right)$ and $P\left(\tilde{C}_i|\mathcal{G}, v_i\right)$ separately.

Unfortunately, exactly inferring of $P\left(y_i|\tilde{C}_i, \mathcal{G}, v_i\right)$ or $P\left(\tilde{C}_i|\mathcal{G}, v_i\right)$ is not feasible. This is because, on graph data, the labels of a node v_i (both $\tilde{C}_i$ and y_i) are determined by its attributed ego net, and exactly inferring them requires enumerating all the possible attributed ego nets. Therefore, we approximate the above two distributions by tractable models, including a classifier $f(\theta)$ and a weight assigner $g(\phi)$. In the following subsections, we will introduce the instantiations of $f(\theta)$ and $g(\phi)$; after that, a novel generalized episodic training paradigm is presented, which is tailored for the generalized few-shot node classification problem and can work hand-in-hand with our models.

Models. The overall framework of our proposed model STAGER is presented in Fig. 3.8. It is composed of a classifier $f(\theta)$ and a weight assigner $g(\phi)$ introduced below.

A - Classifier $f(\theta)$. The classifier is instantiated as follows which is based on the *predict-then-propagate* design [23],

$$\mathbf{H}^{(0)} = \texttt{MLP}\left(\mathbf{X}, \theta\right), \tag{3.23a}$$

$$\mathbf{H}^{(j+1)} = \tilde{\mathbf{A}}\mathbf{H}^{(j)}, \tag{3.23b}$$

$$\mathbf{Z} = \texttt{softmax}\left(\sum_{j=0}^{p}\left(\mathbf{W}[:, j]\mathbf{1}^\top\right) \odot \mathbf{H}^{(j)}\right). \tag{3.23c}$$

We first obtain the prediction $\mathbf{H}^{(0)} \in \mathbb{R}^{n\times C}$ based on the node attributes $\mathbf{X}$ from a multi-layer perceptron (MLP) parameterized by θ (i.e., Eq. (3.23a)), where n is the number of nodes and C is the total number of node classes (including both $\mathcal{C}_{\texttt{base}}$ and $\mathcal{C}_{\texttt{novel}}$). Then, the prediction matrix $\mathbf{H}^{(0)}$ is propagated p steps to obtain a group of prediction matrices $\{\mathbf{H}^{(0)}, \ldots, \mathbf{H}^{(p)}\}$ by power iterations with $\tilde{\mathbf{A}}$ (i.e., Eq. (3.23b)). Here $\tilde{\mathbf{A}} \in \mathbb{R}^{n\times n}$ is the symmetrically normalized adjacency matrix with self-loops. Notice that these prediction matrices also denote the predictions based on different receptive fields. Finally, all these prediction matrices are aggregated by a weight assignment matrix $\mathbf{W} \in \mathbb{R}^{n\times(p+1)}$ whose entry $\mathbf{W}[i, j]$ represents the importance of the j-th propagated prediction matrix (i.e., $\mathbf{H}^{(j)}$) for the i-th node (i.e., $\mathbf{Z}[i, :]$). By broadcasting the weight vector for the j-th propagation (i.e., $\mathbf{W}[:, j]$) with an all-one vector $\mathbf{1} \in \mathbb{R}^{C\times 1}$, we assign the weight to the j-th propagated

prediction matrix through the Hadamard product $\odot$. The $\texttt{softmax}$ is row-wise.

Existing models APPNP [23] and GPRGNN [121] set every column of weight matrix $\mathbf{W}$ as a constant vector, which might suffer from the *inconsistent preference* problem as the optimal weight assignments among receptive fields could vary dramatically between the many-shot cases and the few-shot cases (see the detailed analysis in Section 3.2.1). One possible solution is to set $\mathbf{W}$ as a free learnable parameter of the classifier, which is prone to overfitting as the number of parameters is linear with respect to the number of nodes. More importantly, as we have analyzed before, explicitly estimating $P\left(\tilde{C}_i|\mathcal{G}, v_i\right)$ is necessary. Hence, we propose to encode $P\left(\tilde{C}_i|\mathcal{G}, v_i\right)$ into $\mathbf{W}$ as the output of a well-designed *weight assigner* as introduced below.

B - Weight Assigner $g(\phi)$**.** Based on the design motivation laid out above, on estimating $P\left(\tilde{C}_i|\mathcal{G}, v_i\right)$ and working closely with the classifier $f(\theta)$, our design of the weight assigner $g(\phi)$ bears the following rationales. First, essentially, for the generalized few-shot node classification problem, $P\left(\tilde{C}_i|\mathcal{G}, v_i\right)$ reflects the number of shots; e.g., $P\left(\tilde{C}_i = base|\mathcal{G}, v_i\right)$ indicates the probability of the class of v_i being provided with many shots. Second, a sub-module should explicitly extract the shot-aware representation from the input $\{\mathcal{G}, v_i\}$. Finally, if the input of a module is shot-aware, its output should also be shot-aware. Concrete instantiation of g is as follows whose two sub-modules are parameterized by $\phi = \{\phi_1, \phi_2\}$,

$$\tilde{\mathbf{Z}} = \texttt{rank}\left(\texttt{softmax}\left(g_1\left(\mathbf{A}, \mathbf{X}, \phi_1\right)\right)\right), \tag{3.24a}$$

$$\mathbf{W} = \texttt{MLP}\left(\tilde{\mathbf{Z}}, \phi_2\right). \tag{3.24b}$$

The first sub-module (Eq. (3.24a), motivated by the first and second rationales) is a preliminary node classifier g_1 (parameterized by ϕ_1, followed with $\texttt{softmax}$) whose *ranked* output $\tilde{\mathbf{Z}} \in \mathbb{R}^{n \times C}$ is the shot-aware representation. The second sub-module is an MLP (Eq. (3.24b), parameterized by ϕ_2, following the last rationale), whose output is the shot-aware weight assignment matrix. The $\texttt{rank}$ is row-wise.

Our key idea is to utilize the *epistemic uncertainty*.[1] Since the epistemic uncertainty can reflect the size of training data, i.e., shots for our problem, our first sub-module of the weight assigner (Eq. (3.24a)) conducts a preliminary prediction (i.e., $\texttt{softmax}(g_1(\cdot))$, and then measures the epistemic uncertainty (i.e., $\texttt{MLP}(\texttt{rank}(\cdot))$, where $\texttt{MLP}$ is merged into another $\texttt{MLP}$ from Eq. (3.24b) for brevity) to output the shot-aware representation $\tilde{\mathbf{Z}}$. We further elaborate on a few more points. First, for better extracting the epistemic uncertainty, we adopt a practical approach named dropout variational inference [144, 146] and rewrite

[1]Epistemic uncertainty is defined to measure how well the model fits the data and is reducible as the size of training data increases [144, 145].

the input of $\texttt{rank}(\cdot)$ from $\texttt{softmax}\big(g_1(\mathbf{A}, \mathbf{X}, \phi_1)\big)$ to $\frac{1}{T}\sum_{t=1}^{T}\texttt{softmax}\left(g_1\left(\mathbf{A}, \mathbf{X}, \phi_1^t\right)\right)$ where ϕ_1^t is the masked parameter of ϕ_1 through dropout layers [147]. Second, the design of the $\texttt{rank}$ function is based on the intuition that the uncertainty is closely related to the ordered prediction vector. For example, commonly-used uncertainty metrics of a prediction vector $\mathbf{u}$, $\texttt{max}(\mathbf{u})$ (the largest probability) and $\texttt{gap}(\mathbf{u})$ (the largest probability minus the second largest one) can be represented by $[1, 0, \ldots, 0] \cdot \texttt{rank}(\mathbf{u})$ and $[1, -1, \ldots, 0] \cdot \texttt{rank}(\mathbf{u})$, respectively. Finally, the proposed uncertainty measure: $\texttt{MLP}(\texttt{rank}(\cdot))$ is flexible, thanks to the *universal approximator* $\texttt{MLP}$ [148]. We will provide an interesting discussion about the selection of g_1 in Section 3.2.4.

Imbalanced Episodic Training. As mentioned, we decompose the classification goal into two probability distributions ($P\left(y_i|\tilde{C}_i, \mathcal{G}, v_i\right)$ and $P\left(\tilde{C}_i|\mathcal{G}, v_i\right)$) and estimate them by the classifier $f(\theta)$ and weight assigner $g(\phi)$ respectively. The key idea is designing $g(\phi)$ to output a shot-aware weight assignment matrix for the $f(\theta)$. This is because, fundamentally, for the generalized few-shot node classification problem, $P\left(\tilde{C}_i|\mathcal{G}, v_i\right)$ reflects shots and vice versa. Here, we further ask: *how can we train $g(\phi)$ to estimate $P\left(\tilde{C}_i|\mathcal{G}, v_i\right)$ in an even broader scope, beyond the scenario of base classes $\mathcal{C}_{base}$ vs. novel classes $\mathcal{C}_{novel}$?* In other words, whether there exist other base classes vs. novel classes scenarios from which the $g(\phi)$ can learn?

To answer this question, let us take a close look at a prevalent training paradigm for meta-learning problems named episodic training [124, 125, 130, 131, 132]. The core idea of episodic training is to leverage the abundant labeled base samples to generate sufficient few-shot episodes (all classes are few-shot). The few-shot episodes facilitate the learning of a meta-learner which can in turn assist the learning of learners on few-shot scenarios. However, directly grafting such a strategy on the training of $g(\phi)$ is pointless. That is because, the goal of the weight assigner ($g(\phi)$) for the generalized few-shot node classification problem is not *learning to learn a few-shot classifier* (the core idea of meta-learning based few-shot learning), but *learning to tell if a node is from novel classes or base classes*. Based on this key insight, we generalize the episodic training and propose a novel training strategy named *imbalanced episodic training*.

A - Imbalanced Episodes. For the generalized few-shot node classification task, as we have mentioned before, the ideal training scenarios of the weight assigner $g(\phi)$ are composed of base classes vs. novel classes. Hence, we propose *imbalanced episodic training* to mimic such scenarios. Particularly, our first step is to sample *pseudo-base* and *pseudo-novel* classes ($\mathcal{C}_{\texttt{pseudo-novel}}$ and $\mathcal{C}_{\texttt{pseudo-base}}$) from the base classes such that $|\mathcal{C}_{\texttt{pseudo-novel}}| = N$ and

$|\mathcal{C}_{\texttt{pseudo-base}}| = M$. Then, the *labelled nodes (from $\mathcal{C}_{base}$)* which belongs to $\mathcal{C}_{\texttt{pseudo-novel}}$ and $\mathcal{C}_{\texttt{pseudo-base}}$ are notated as $\mathcal{V}_{\texttt{pseudo-novel}}$ and $\mathcal{V}_{\texttt{pseudo-base}}$. After that, episodes $\{\mathcal{E}_i = \{\mathcal{S}_i, \mathcal{Q}_i\}\}$ are sampled from $\mathcal{V}_{\texttt{pseudo-novel}}$ and $\mathcal{V}_{\texttt{pseudo-base}}$ where the i-th episode can be represented as follows.

$$\begin{aligned} \mathcal{S}_i &= \{\underbrace{v_1, \ldots, v_{N\times K}}_{\text{from } \mathcal{V}_{\texttt{pseudo-novel}}}, \underbrace{v_{N\times K+1}, \ldots, v_{N\times K+M\times L}}_{\text{from } \mathcal{V}_{\texttt{pseudo-base}}}\} \\ \mathcal{Q}_i &= \{\underbrace{v'_1, \ldots, v'_{N\times I}}_{\text{from } \mathcal{V}_{\texttt{pseudo-novel}}}, \underbrace{v'_{N\times I+1}, \ldots, v'_{(N+M)\times I}}_{\text{from } \mathcal{V}_{\texttt{pseudo-base}}}\} \end{aligned} \tag{3.25}$$

where K nodes are sampled from $\mathcal{V}_{\texttt{pseudo-novel}}$ per class and L nodes are sampled from $\mathcal{V}_{\texttt{pseudo-base}}$ per class to form every support set $\mathcal{S}$; from both $\mathcal{V}_{\texttt{pseudo-base}}$ and $\mathcal{V}_{\texttt{pseudo-novel}}$, I nodes are sampled per class to form every query set $\mathcal{Q}$. About the selection of N, M, K, L, I, we follow three rules of thumb: (1) $N \ll M$ because it is common that $|\mathcal{C}_{\texttt{novel}}| \ll |\mathcal{C}_{\texttt{base}}|$, (2) $K \ll L$ because $\mathcal{C}_{\texttt{pseudo-base}}$ serves as the many-shot classes and $\mathcal{C}_{\texttt{pseudo-novel}}$ serves as the few-shot classes, and (3) I is fixed (e.g., 30) as many existing works [124, 131] did. Specific selections of the above values can be found in Section 3.2.4.

B - Training Procedure for Stager. We rewrite the final prediction of the proposed Stager from Eq. (3.23c) as $z(\mathcal{G}, \theta, \phi_1, \phi_2, v_i)$ to represent the prediction results w.r.t. the i-th node. Following the same format, we rewrite the prediction from the preliminary predictor g_1 (i.e., $\texttt{softmax}\big(g_1(\mathbf{A}, \mathbf{X}, \phi_1)\big)$ from Eq. (3.24a)) as $\tilde{z}(\mathcal{G}, \phi_1, v_i)$. In addition, we use y_i to represent the label of the i-th node. Then the objective of imbalanced episodic training for Stager is,

$$\begin{aligned} \phi_2^* &= \arg\min_{\phi_2} \ \mathbb{E}_{v_i \in \mathcal{Q}} \ L_{\texttt{cla}}\left(z\left(\mathcal{G}, \theta^*, \phi_1^*, \phi_2, v_i\right), y_i\right), \\ s.t.\ \theta^*, \phi_1^* &= \arg\min_{\theta, \phi_1} \ \mathbb{E}_{v_j \in \mathcal{S}} \ L_{\texttt{cla}}\left(z\left(\mathcal{G}, \theta, \phi_1, \phi_2, v_j\right), y_j\right) + \lambda L_{\texttt{cla}}\left(\tilde{z}\left(\mathcal{G}, \phi_1, v_j\right), y_j\right), \end{aligned} \tag{3.26}$$

where λ is a trade-off parameter and $L_{\texttt{cla}}$ denotes the classification loss.

Remarks. First, in Eq. (3.26), the episodes are imbalanced according to Eq. (3.25). Second, the preliminary classifier $g_1(\phi_1)$ should be trained from scratch to be shot-aware in every episode; hence, ϕ_1 is optimized in the lower-level objective and only ϕ_2 is updated across episodes. Finally, in implementation, for every episode, we pretrain the preliminary classifier $g_1(\phi_1)$ based on $L_{\texttt{cla}}(\tilde{z}(\mathcal{G}, \phi_1, v_j), y_j)$ (which does not contain θ or ϕ_2) to converge and keep it fixed when solving the bilevel optimization problem in Eq. (3.21) (i.e., remove the term $\lambda L_{\texttt{cla}}(\tilde{z}(\mathcal{G}, \phi_1, v_j), y_j)$ from the lower-level objective). Such a training strategy shows great efficacy empirically. In principle, we can use any gradient descent-based optimization. To

Algorithm 3.2: Imbalanced episodic training for STAGER

Input : an attributed graph $\mathcal{G} = \{\mathbf{A}, \mathbf{X}\}$, the hyper-parameters of the imbalanced episode N, M, K, L, I;
Output: optimized ϕ_2;

initialize ϕ_2;
while ϕ_2 *not converged* **do**
  construct an imbalanced episode $\{\mathcal{E} = \{\mathcal{S}, \mathcal{Q}\}\}$ according to Eq. (3.25);
  pretrain ϕ_1 based on the classification loss given $\mathcal{S}$;
  initialize θ;
  while θ *not converged* **do**
    update θ based on the classification loss given $\mathcal{S}$;
  end
  update ϕ_2 based on the classification loss given $\mathcal{Q}$;
end

compute the gradient of the upper-level problem, many choices such as the first-order approximation [117, 149] and iterative differentiation methods [150, 151] are available. At the meta-test phase, we train the classifier $f(\theta)$ and preliminary classifier $g_1(\phi_1)$ from scratch and fine-tune the second module of weight assigner parameterized with ϕ_2 on all the labeled nodes with sample re-weighting.

The training procedure is formally presented in Algorithm 3.2.

C - Discussion. If the nodes notated as 'from $\mathcal{V}_{\texttt{pseudo-base}}$' are removed in Eq. (3.25), the composition of both the support set $\mathcal{S}_i$ and the query set $\mathcal{Q}_i$ is the same as Eq. (3.20). From another perspective, the conventional episodic training mimics the uniform distribution of the number of nodes from the selected N (pseudo-novel) classes under the few-shot settings. Our imbalanced episodic training takes one step further and mimics a more complex mixed distribution from two uniform distributions (i.e., many shots for $\mathcal{C}_{\texttt{pseudo-base}}$ and few shots for $\mathcal{C}_{\texttt{pseudo-novel}}$).

Importantly, this training paradigm is independent of our model STAGER, and it is even independent of the node classification task. In fact, it is a new *general* meta-training paradigm which can be applied to most, if not all, of the generalized few-shot learning problem from the meta-learning perspective. In addition, the idea behind the proposed imbalanced episode training can be naturally generalized to other more realistic distributions such as the long-tailed power-law distribution. We leave those interesting topics as future works.

We noticed that some existing efforts [60, 152, 153, 154] generalize the traditional episodic training into a noisy variant (e.g., the shot per class in the support set is a random number). We elaborate on the similarity and differences between the noisy episodic training

and our proposed imbalanced episodic training as follows. Both training strategies aim to mimic a realistic training episode, where the training samples per class are not the same. Our proposed imbalanced episodic training aims to emphasize the comparison between the pseudo-based classes vs. the pseudo-novel classes (e.g., 50 shots vs. 5 shots) so as to empower the training of the uncertainty-aware module. However, the noisy episodic training does not have a specific focus on such a comparison. A potential improvement of the imbalanced episodic training is to manually import some noise into the number of training samples. For example, the shots of pseudo-base classes can be $\texttt{round}(50 + \texttt{noise})$ and the shots of pseudo-novel classes can be $\texttt{round}(5 + \texttt{noise})$. We also leave those interesting explorations as future works.

For the model complexity, compared with other models following the *predict-then-propagate* design [23] (e.g., APPNP [23] and GPRGNN [121]), the proposed STAGER needs two extra models ϕ_1 and ϕ_2 from Eq. (3.24a) and Eq. (3.24b). For brevity, if both ϕ_1 and ϕ_2 have only one hidden layer and the hidden feature dimension is the same as the raw feature dimension d, the number of parameters of ϕ_1 and ϕ_2 are $d^2|\mathcal{C}|$ and $d|\mathcal{C}|(p+1)$, respectively. Here $\mathcal{C}$ is the node class set and p is the propagation steps.

3.2.4 Experiments

Datasets. We use two E-commerce datasets: Amazon Clothing[2] [136], Amazon Electronics[2] [136], and two citation datasets: Aminer[3] [155], and Cora-Full[4] [137]. Detailed statistics of the datasets is in Table 3.8. All the datasets used in this paper are publicly accessible. They are all anonymized, numerized, and do not contain personally identifiable information or offensive content. For all the datasets, we only select classes whose number of nodes is larger than or equal to 100, so that we can have sufficient test nodes for every class. For the **novel classes**, we follow the N-way K-shot setup where $N \in \{5, 10\}$ and $K \in \{1, 3\}$. Also, we select **many base classes** (i.e. $N \ll |\mathcal{C}_{\texttt{base}}|$) with 50 shots and select a part of the base classes as the validation classes. Note that our setting is also known as the step imbalance [156], where the novel classes are with k shots and the base classes are with 50 shots. We leave studies on other types of imbalance as future work. Our code and data are accessible [5] with detailed dataset splits and hyperparameter settings.

[2]https://nijianmo.github.io/amazon/index.html
[3]https://www.aminer.cn/data/?nav=openData
[4]https://github.com/abojchevski/graph2gauss/tree/master/data
[5]https://github.com/pricexu/STAGER

Table 3.8: Statistics of Datasets.

Dataset	Nodes	Edges	Features	Labels
Amazon Clothing	24919	91680	9034	77
Amazon Electronics	42318	43556	8669	167
Aminer	40672	288270	7202	137
Cora-Full	18800	62685	8710	56

Metrics. The performance of models is evaluated by the accuracy (ACC) on test nodes. To be specific, we report the accuracy on the base classes, novel classes, and all the classes, respectively. We report the average result, along with the standard deviation, in 10 runs.

Baseline Methods. The baseline methods which we compare our STAGER-I (with imbalanced episodic training) and STAGER (without imbalanced episodic training) with can be categorized into (1) classic neural node classifiers including APPNP [23], and GPRGNN [121] which follow the same predict-then-propagate design as our STAGER; (2) few-shot neural node classifiers including MetaGNN [125], GPN [124], and G-META [127]; (3) an imbalanced node classifier GraphSMOTE [157] (short as G-SMOTE). Note that if we ablate the weight assigner g from the proposed STAGER, our model will degenerate into the GPRGNN [121] whose weights of receptive fields are the same for every node.

Implementation. For the APPNP and GPRGNN, we train them with two strategies: (1) pre-training models over the base classes and fine-tuning them over the novel classes or 'novel & base' classes, and (2) training models over the imbalanced labeled nodes and re-weighting nodes from the novel classes with high weights. We empirically find the performance from the second strategy is better and we report their best performance in the following subsections. For MetaGNN, GPN, and G-META, at the meta-training phase, we train them with the existing episodic training. At the meta-test phase, we fine-tune MetaGNN on the imbalanced labeled nodes. For GPN and G-META, at the meta-test phase, since their models and codes are designed for the balanced few-shot settings, we downsample the labeled nodes from base classes so that all the classes are few-shot. For GraphSMOTE, we implement its downstream classifier as APPNP which shows strong performance. As GraphSMOTE conducts node augmentation based on the nodes from novel classes only, when the shot is 1 for the novel classes, there is no augmentation space and we report the same results as APPNP. When shots are 3, GraphSMOTE augments novel classes first and then trains APPNP over the augmented data.

For the proposed STAGER, θ, ϕ_1, and ϕ_2 share the same number of hidden units which is

searched from $\{24, 48, 72, 96, 120\}$. We train the model with Adam [158]. The learning rate is searched from $\{1\times10^{-2}, 5\times10^{-3}, 1\times10^{-3}, 5\times10^{-4}\}$ and the weight decay is searched from $\{1\times10^{-4}, 5\times10^{-4}, 1\times10^{-3}, 5\times10^{-3}, 1\times10^{-2}, 5\times10^{-2}, 0.1\}$. The sample re-weighting is set as $50 : N$ for the novel and base classes respectively. N is the number of ways for the novel classes. The integer number of steps for propagation p is searched from $[2, 10]$. The dropout rate of the classifier f and the weight assigner g is searched from $\{0, 0.2, 0.4, 0.5, 0.6, 0.8\}$. For the imbalanced episodic training, the number of query nodes per class $I = 30$; the number of pseudo-novel classes N and the number of shots per pseudo-novel class K follow the specific N-way K-shot setup; the number of pseudo-base classes M is the number of all the remaining base classes (i.e., $M = |\mathcal{C}_{\texttt{base}}| - N$); the number of shots per pseudo-base class $L = 40$. We set the dropout variational inference parameter $T = 10$, as we observed that when $T \geq 10$, model performance becomes very stable. Our code and data are accessible online[6].

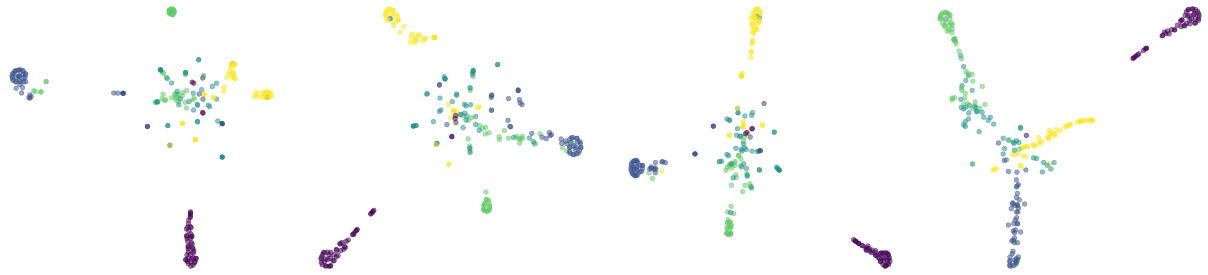
Main Results. Performance comparison on four datasets is presented in Table 3.9. We have the following observations. First, existing few-shot node classifiers do not perform well on the generalized few-shot node classification problem. For instance, MetaGNN does not show advantages compared with classic methods such as APPNP; GPN and G-META can obtain a decent performance on the novel classes but cannot fully utilize the labeled nodes from the base classes during the meta-test phase, which in turn degrades its performance on the base classes. Second, compared with classic neural node classifiers (APPNP, GPRGNN), without imbalanced episodic training, the proposed STAGER already outperforms them in most cases and retains competitive in the remaining cases. Third, there is a trade-off between the performance on the base and novel classes and we observe that, in most cases, the proposed imbalanced episodic training can indeed significantly improve the performance on the novel classes while keeping competitive performance on the base classes. Fourth, in all the cases, our models (STAGER and STAGER-I) obtain the *best overall performance* on all the settings consistently and significant improvements on some cases (e.g., 13.3% improvement on all the classes on Amazon Electronic datasets with the 5-way 3-shot setting). Finally, GraphSMOTE only leverages the novel classes to conduct node augmentation whose advantage is restricted.

In addition, we provide the visualization of the prediction vectors of test nodes from the novel classes in Figure 3.9. GPRGNN, G-META, GraphSMOTE, and STAGER-I are selected for the visualization and the experimental setting is selected as 5-way 3-shot on the Amazon Clothing dataset. The visualization shows that STAGER-I can effectively discriminate novel

[6]https://github.com/pricexu/STAGER

Table 3.9: Performance comparison (mean±std accuracy %) on four datasets under different N-way K-shot settings for *novel classes*. 'Base', 'Novel', and 'All' denote accuracies on the base classes, novel classes, and the entire label space, respectively. Bold and underlined numbers indicate the best and second-best performance, respectively.

Dataset	Setting	Class	APPNP	GPRGNN	MetaGNN	GPN	G-META	G-SMOTE	STAGER	STAGER-I
Amazon Clothing	5w1s	Base	67.4±1.6	64.7±0.5	64.0±0.5	46.1±3.3	48.7±2.7	67.4±1.6	**69.3±1.1**	67.3±0.4
		Novel	31.4±0.9	31.5±5.4	28.3±0.4	36.1±5.1	39.2±2.9	31.4±0.9	32.4±2.0	**41.3±1.0**
		All	48.5±1.0	47.3±3.0	45.4±0.4	40.9±2.9	43.7±2.2	48.5±1.0	50.0±1.0	**53.7±0.5**
	5w3s	Base	70.5±0.9	69.7±1.0	66.1±1.2	62.9±1.9	63.3±1.7	69.4±0.7	**72.3±1.5**	68.4±1.0
		Novel	48.6±2.5	50.1±3.8	40.4±0.8	46.1±6.7	47.6±6.3	45.6±2.3	53.9±2.1	**66.0±2.4**
		All	59.1±1.2	59.4±2.1	52.6±1.0	54.1±3.4	55.1±3.5	57.0±1.4	62.7±1.1	**67.2±1.1**
	10w1s	Base	73.3±0.3	70.7±1.2	67.6±0.5	42.7±2.4	48.2±2.1	73.3±0.3	**76.7±1.5**	66.7±0.5
		Novel	45.2±0.6	37.7±3.1	41.5±0.5	39.7±5.7	39.9±4.9	45.2±0.6	43.1±2.7	**59.6±0.6**
		All	58.6±0.3	53.5±1.6	54.0±0.4	40.9±3.5	43.9±2.0	58.6±0.3	59.0±1.6	**63.0±0.5**
	10w3s	Base	69.2±0.6	67.5±1.1	65.6±1.2	59.5±2.7	57.4±1.9	68.1±0.7	**70.9±0.7**	69.3±0.4
		Novel	61.4±0.4	58.0±1.5	53.6±0.2	49.6±7.1	54.1±2.8	53.6±3.8	61.8±1.2	**64.6±0.7**
		All	65.2±0.4	62.5±1.3	59.2±0.2	54.3±3.4	55.6±1.6	60.5±1.9	66.2±0.8	**66.8±0.5**
Amazon Elec.	5w1s	Base	60.1±1.8	58.4±0.9	59.7±0.3	19.1±2.1	22.5±3.1	60.1±1.8	**65.8±2.1**	63.9±1.0
		Novel	7.8±0.8	5.1±1.1	6.4±0.3	16.6±5.4	15.3±6.7	7.8±0.8	8.0±0.7	**19.7±1.6**
		All	27.2±0.4	24.8±0.4	26.2±0.2	17.5±3.8	18.0±5.0	27.2±0.4	29.4±1.4	**36.1±1.1**
	5w3s	Base	64.2±1.8	55.1±0.9	63.0±0.7	43.7±1.6	43.6±2.4	63.0±1.4	**69.1±1.6**	69.0±2.9
		Novel	21.6±1.5	13.3±2.0	23.1±0.2	32.7±4.8	28.1±5.6	12.0±4.0	29.8±2.7	**40.7±2.2**
		All	37.4±1.5	28.8±1.4	37.9±0.3	36.8±3.4	33.9±3.5	30.9±2.5	44.3±1.8	**51.2±2.3**
	10w1s	Base	64.4±1.2	59.7±1.3	53.1±1.6	18.5±1.4	20.8±2.2	64.4±1.2	**69.0±0.9**	61.3±0.8
		Novel	8.0±1.3	5.7±1.1	4.9±0.1	15.3±3.7	15.0±3.7	8.0±1.3	11.3±1.5	**15.4±0.3**
		All	34.4±1.0	31.0±1.1	27.7±0.2	16.8±2.3	17.7±2.0	34.4±1.0	**38.3±1.2**	36.9±0.4
	10w3s	Base	58.6±0.4	55.2±0.9	48.8±0.7	43.8±1.7	46.3±1.6	62.9±0.7	**72.3±1.1**	66.5±0.1
		Novel	22.4±1.1	14.8±1.0	16.5±0.2	27.5±2.9	26.2±3.2	13.8±0.3	20.3±2.4	**38.1±2.3**
		All	39.4±0.5	33.7±0.7	31.6±0.4	35.1±1.4	35.6±1.8	36.8±0.2	44.7±1.5	**51.4±1.1**
Aminer	5w1s	Base	40.8±1.0	38.2±1.7	40.4±0.4	19.4±1.5	25.4±1.7	40.8±1.0	**40.9±1.0**	36.1±1.1
		Novel	24.8±2.3	12.4±2.3	7.6±0.2	20.0±6.4	22.2±4.6	24.8±2.3	29.7±1.2	**37.2±1.6**
		All	32.5±1.3	24.8±1.1	23.5±0.3	19.7±3.1	23.7±2.6	32.5±1.3	35.7±0.6	**36.7±1.3**
	5w3s	Base	42.5±1.6	39.8±1.5	**42.6±0.4**	23.0±1.9	38.2±1.4	39.2±1.5	42.0±1.1	39.6±0.2
		Novel	33.1±0.9	29.3±2.5	33.4±0.3	21.4±4.2	34.9±4.4	36.3±1.3	36.2±0.8	**44.4±0.3**
		All	37.6±0.9	34.4±0.8	37.8±0.2	22.2±2.1	36.5±2.4	37.7±0.4	39.0±0.9	**42.1±0.3**
	10w1s	Base	41.2±1.3	41.0±0.8	**42.6±0.4**	19.6±1.5	23.4±1.8	41.2±1.3	40.3±1.6	40.4±0.4
		Novel	11.2±1.2	4.1±1.3	11.6±0.3	16.1±4.0	15.5±4.4	11.2±1.2	16.8±0.6	**21.8±1.1**
		All	25.7±0.6	22.0±0.7	26.6±0.2	17.8±2.2	19.3±2.3	25.7±0.6	28.2±1.1	**30.7±0.5**
	10w3s	Base	41.7±1.2	43.1±1.5	42.4±0.2	26.3±1.5	34.7±1.5	39.1±0.5	**46.2±0.8**	40.4±0.6
		Novel	21.6±0.3	17.7±1.8	24.5±0.3	18.7±3.8	23.4±4.8	23.0±1.4	21.6±0.6	**27.8±1.0**
		All	31.3±0.6	30.0±1.1	33.1±0.2	22.4±2.2	28.9±2.1	30.8±0.9	33.5±0.6	**33.9±0.7**
Cora Full	5w1s	Base	70.6±1.0	**72.7±1.3**	71.4±0.2	40.2±3.1	55.5±2.7	70.6±1.0	71.1±0.8	70.1±2.1
		Novel	25.3±0.8	21.3±0.9	20.8±0.3	12.9±4.7	19.0±6.6	25.3±0.8	33.2±1.4	**34.8±0.4**
		All	46.3±0.7	45.1±0.9	44.3±0.2	25.6±3.2	35.9±3.3	46.3±0.7	50.8±1.1	**51.2±0.8**
	5w3s	Base	73.3±1.1	76.5±1.3	70.1±0.2	44.6±2.0	55.5±1.5	73.0±0.9	70.8±0.4	**78.1±0.2**
		Novel	26.5±1.4	21.1±2.4	23.6±0.6	16.1±2.7	24.8±4.4	9.7±0.7	**32.0±2.3**	30.7±2.3
		All	48.2±1.1	46.8±1.7	45.2±0.3	29.3±1.6	39.1±2.5	39.1±0.3	50.0±1.3	**52.6±1.4**
	10w1s	Base	75.9±0.8	**76.2±0.8**	70.1±0.2	45.2±3.0	52.1±1.7	75.9±0.8	**76.2±0.7**	69.6±0.1
		Novel	20.1±1.7	15.1±1.3	11.5±0.2	14.6±2.3	16.9±3.1	20.1±1.7	**29.8±0.9**	24.3±0.5
		All	40.7±0.7	44.6±0.8	39.8±0.1	29.4±1.4	33.9±1.4	40.7±0.7	**52.2±0.1**	46.2±0.3
	10w3s	Base	74.8±0.6	74.7±0.6	68.0±0.3	42.8±1.5	52.7±1.8	67.5±0.5	**77.9±0.6**	74.4±0.4
		Novel	37.9±1.4	28.1±1.2	34.1±0.3	19.4±2.9	18.6±2.9	7.5±1.0	27.7±2.0	**40.0±1.4**
		All	55.7±0.7	50.7±0.7	50.5±0.2	30.7±1.2	35.0±1.3	36.5±0.7	52.0±0.7	**56.6±0.6**



(a) GPRGNN (b) G-META (c) GraphSMOTE (d) STAGER-I

Figure 3.9: Visualization of predictions from GPRGNN (a), G-META (b), GraphSMOTE (c), and STAGER-I (d). Best viewed in color.

test nodes, which aligns well with the experimental results from Table 3.9.

Ablation Study: Weight Assigner. The weight assigner g is the key component of STAGER. If we remove the weight assigner and directly set the weights assigned to receptive fields as trainable parameters, our model will degenerate to GPRGNN [121]. Hence, we study the performance comparison between GPRGNN [121] and STAGER(without imbalanced episodic training) for the 10-way 3-shot setup and present the accuracy on the novel classes and all the classes in Table 3.10a. Notice that the data is re-organized from Table 3.9 where Amazon-C and Amazon-E represent Amazon Clothing and Amazon Electronics datasets, respectively. Clearly, we observe that with the weight assigner, in most cases, the accuracy on novel classes and all the classes gets boosted significantly.

Ablation Study: Rank Operator. We stated in the main content that the rank operator is designed to extract uncertainty from the prediction vectors. Here, we study the performance comparison between STAGER-I (without rank operator) and STAGER-I (with rank operator) for the 10-way 3-shot setup and present the accuracy on the novel classes and all the classes in Table 3.10b where Amazon-C and Amazon-E represent Amazon Clothing and Amazon Electronics datasets, respectively. Clearly, we observe that the rank operator, in most cases, can significantly boost the accuracy on novel classes and all classes.

Ablation Study: Imbalanced Episodic Training. To study the effectiveness of imbalanced episodic training, first, we study the performance comparison between STAGER (without imbalanced episodic training) and STAGER-I (with imbalanced episodic training) for the 10-way 3-shot setup. The accuracy on the novel classes and all the classes is presented in Table 3.11a and this part of the data is re-organized from Table 3.9; then we implement imbalanced episodic training on MetaGNN (notated as MetaGNN-I) for the 10-way 3-shot

Table 3.10: Ablation study on the weight assigner g (a) and the rank operator (b) (mean±std accuracy (%)). The number in parentheses indicates the performance comparison with the ablated variants in the left columns.

(a) Weight assigner g

Dataset	Stager (without g)		Stager (with g)	
	Novel	All	Novel	All
Amazon-C	58.0±1.5	62.5±1.3	61.8±1.2 (+3.8)	66.2±0.8 (+3.7)
Amazon-E	14.8±1.0	33.7±0.7	20.3±2.4 (+5.5)	44.7±1.5 (+11.0)
Aminer	16.1±1.7	28.2±1.1	21.6±0.6 (+5.5)	33.5±0.6 (+5.3)
Cora Full	28.1±1.2	50.7±0.7	27.7±2.0 (-0.4)	52.0±0.7 (+1.3)

(b) Rank operator

Dataset	Stager-I (without rank)		Stager-I (with rank)	
	Novel	All	Novel	All
Amazon-C	57.8±1.1	61.7±0.7	64.6±0.7 (+6.8)	66.8±0.5 (+5.1)
Amazon-E	20.6±1.5	43.2±1.0	38.1±2.3 (+17.5)	51.4±1.1 (+8.2)
Aminer	23.6±0.9	35.9±0.7	27.8±1.0 (+4.2)	33.9±0.7 (-2.0)
Cora Full	34.8±0.9	49.1±0.8	40.0±1.4 (+5.2)	56.6±0.6 (+7.5)

Table 3.11: Ablation study on the imbalanced episodic training for Stager (a) and MetaGNN (b) (mean±std accuracy (%)). The number in parentheses indicates the performance comparison with the ablated variants in the left columns.

(a) Imbalanced episodic training for Stager

Dataset	Stager		Stager-I	
	Novel	All	Novel	All
Amazon-C	61.8±1.2	66.2±0.8	64.6±0.7 (+2.8)	66.8±0.5 (+0.6)
Amazon-E	20.3±2.4	44.7±1.5	38.1±2.3 (+17.8)	51.4±1.1 (+6.7)
Aminer	21.6±0.6	33.5±0.6	27.8±1.0 (+6.2)	33.9±0.7 (+0.4)
Cora Full	27.7±2.0	52.0±0.7	40.0±1.4 (+12.3)	56.6±0.6 (+4.6)

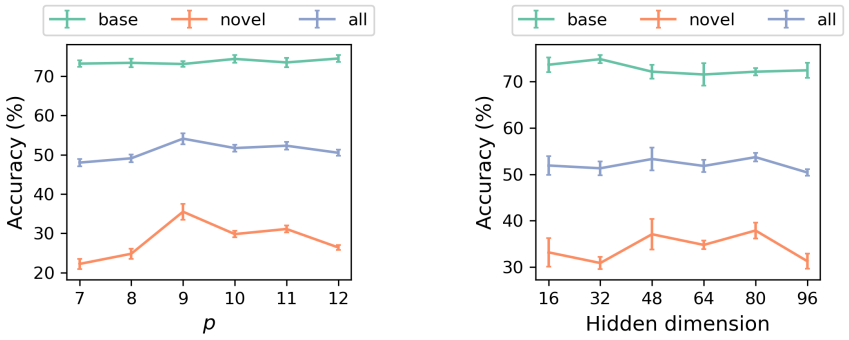
(b) Imbalanced episodic training for MetaGNN

Dataset	MetaGNN		MetaGNN-I	
	Novel	All	Novel	All
Amazon-C	53.6±0.2	59.2±0.2	56.4±0.3 (+2.8)	61.9±0.4 (+2.7)
Amazon-E	16.5±0.2	31.6±0.4	22.7±0.4 (+6.2)	42.5±0.6 (+10.9)
Aminer	40.2±0.5	38.1±0.2	36.5±0.8 (-3.7)	37.0±0.7 (-1.1)
Cora Full	34.1±0.3	50.5±0.2	33.1±0.5 (-1.0)	40.0±0.8 (-10.5)

setup to see the performance comparison on the novel classes and all the classes in Table 3.11b. Amazon-C and Amazon-E represent Amazon Clothing and Amazon Electronics datasets, respectively.

We observe that for Stager-I, compared with Stager, in most cases, accuracy on the novel classes and that on all the classes gets improved, which demonstrates the effectiveness of imbalanced episodic training. For MetaGNN-I we observe that the performance improvement is not stable and may even hurt the performance dramatically. The key reason is that existing few-shot node classifiers are designed for balanced scenarios, and are not able to address the imbalanced distribution of shots between the base classes and the novel classes. Hence, importing imbalanced shot distribution into the meta-training phase may even affect the learning of existing few-shot node classifiers. We also tried to implement imbalanced episodic training on GPN [124] and G-META [127] but they suffer from the out-of-memory problem due to the great number of labeled base nodes so we do not report them here.

Hyperparameter Sensitivity Study. We conduct a hyperparameter sensitivity study on two hyperparameters. The first is the steps of propagation p from Eq. (3.23c), which controls the size of the set of prediction matrices $\{\mathbf{H}^{(0)}, \ldots, \mathbf{H}^{(p)}\}$. We search p from $[7, 12]$ and present the accuracy of the base, novel, and all classes on the Cora-full dataset. Second, we study the



(a) p (b) hidden dimension

Figure 3.10: Sensitivity study of the propagation steps p (a) and the hidden dimension (b).

sensitivity of STAGER's performance (on the Cora-full dataset) with respect to the hidden representation dimension. We search the hidden dimension in $\{16, 32, 48, 64, 80, 96\}$. From Fig. 3.10, we observe that in general, the performance of STAGER is stable regarding p, and when $p = 9$, our model obtains the best performance. Also, the performance of STAGERis stable with respect to the model's hidden dimension.

Efficiency Study. In this section, we study the efficiency of the proposed model STAGER with the imbalanced episodic training. We use the wall-clock time for updating the model in an episode as the metric. Meta-learning-based methods (MetaGNN [125], GPN [124], and G-META [127]) are selected as baseline methods. For the novel classes, they follow the 10-way 3-shot setting. All efficiency study results are from a single NVIDIA Tesla V100 SXM2-32GB GPU on a server with 96 Intel(R) Xeon(R) Gold 6240R CPUs at 2.40GHz and 1.5 TB RAM. Table 3.12 presents the detailed time comparison on four datasets. We have the following observations. First, GPN [124] and G-META [127] are the fastest, which is expected. That is because their core idea is prototype learning [131], so their algorithms are single-loop (as opposed to a bilevel optimization problem). Second, our proposed STAGER is slower than MetaGNN [125]. That is because our model structure (i.e., weight assigner g + multiple-step classifier f) is more complex than the model used by MetaGNN (i.e., SGC [22]). Third, in general, the proposed STAGER can finish the imbalanced episodic training efficiently (e.g., about 30 minutes for 1000 episodes).

Case Study: Uncertainty vs. Propagation. In this case study, we design experiments to answer two research questions: (1) whether there exists a general epistemic uncertainty gap between the many-shot classes and few-shot classes (beyond novel vs. base classes)? (2) How to select g_1 (i.e., the preliminary classifier for weight assigner g)?

Table 3.12: Efficiency comparison (second/episode) with the baseline methods.

Dataset	MetaGNN	GPN	G-Meta	Stager-I
Amazon-C	0.46	0.39	0.42	1.33
Amazon-E	0.64	0.31	0.36	2.21
Aminer	0.58	0.31	0.40	2.26
Cora Full	0.30	0.24	0.30	0.90

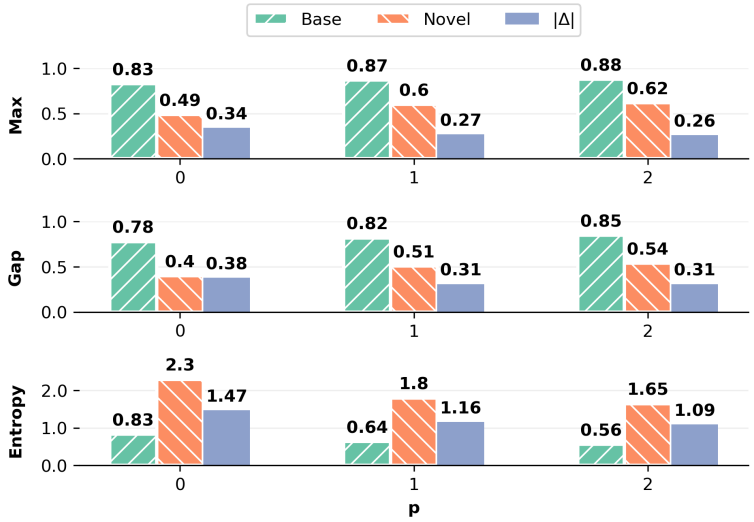
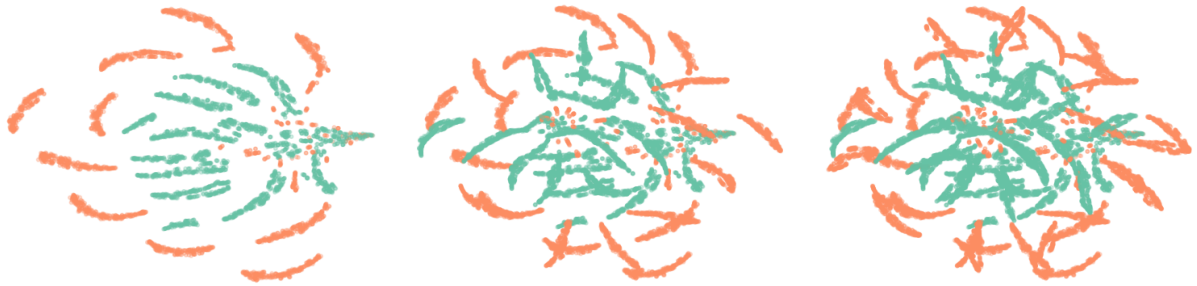


Figure 3.11: Uncertainty comparison between classifiers with different propagation steps. For 'Max' and 'Gap', the smaller the more uncertain. For 'Entropy', the larger the more uncertain. Our goal is to select p where $|\Delta|$ is the largest.



(a) $\texttt{MLP}(\mathbf{X})$ (b) $\texttt{MLP}(\tilde{\mathbf{A}}\mathbf{X})$ (c) $\texttt{MLP}(\tilde{\mathbf{A}}\tilde{\mathbf{A}}\mathbf{X})$

Figure 3.12: Visualization of predictions based on attributes from different propagation. Green points and orange points are predictions of novel classes and base classes respectively. Best viewed in color.

We use the following three metrics to measure the uncertainty in prediction vector $\mathbf{z}$: (1) Max: the largest entry of $\mathbf{z}$; (2) Gap: the gap between the largest entry and the second largest entry of $\mathbf{z}$; (3) Entropy: $-\sum_i \mathbf{z}[i] \log(\mathbf{z}[i])$. To clearly illustrate the influence of

propagation on the prediction uncertainty, we select three direct models: (1) $\texttt{MLP}(\mathbf{X})$, (2) $\texttt{MLP}(\tilde{\mathbf{A}}\mathbf{X})$, (3) $\texttt{MLP}(\tilde{\mathbf{A}}\tilde{\mathbf{A}}\mathbf{X})$, where $\mathbf{X}$ is the node feature matrix and $\tilde{\mathbf{A}}$ is symmetrically normalized adjacency matrix. The `softmax` is applied to normalize the predictions. To measure the epistemic uncertainty, we adopt the dropout variational inference [144, 146] as introduced in Section 3.2.3.

Our experiment design is as follows. First, 50 nodes are randomly selected from every class as test nodes. Second, half of the classes are randomly selected as the novel classes and the remaining classes are the base classes. The training nodes are composed of 1 node per novel class and 50 nodes per base class. Third, the aforementioned three models are trained on the training nodes, and their prediction uncertainty on test nodes is evaluated by the three metrics (i.e., Max, Gap, Entropy) with the dropout variational inference. We repeat the above steps in 10 runs to report the average uncertainties on novel and base classes respectively. Notice that in every run, *the base/novel split is different*.

The experimental results are shown in Fig. 3.11 where x-axis represents models $\texttt{MLP}(\mathbf{X})$, $\texttt{MLP}(\tilde{\mathbf{A}}\mathbf{X})$, and $\texttt{MLP}(\tilde{\mathbf{A}}\tilde{\mathbf{A}}\mathbf{X})$ respectively (i.e., the propagation step is $p = 0$, $p = 1$ and $p = 2$ respectively). $|\Delta|$ denotes the average uncertainty gap between base and novel classes. We have the following observations. First, generally, the uncertainty on the novel classes is much higher than that on the base classes. The above finding directly motivates our model design to set the uncertainty-related module (i.e., ϕ_2 in the weight assigner g) as a meta-learner and learn high-level knowledge (i.e., knowledge can be extracted from the broad base classes by constructing imbalanced episodes) to empower the shot-aware node classifier STAGER. Second, with the increment of propagation steps, the classifier (i.e. `MLP`) is less uncertain on both the novel and base classes. Importantly, the uncertainty gap $|\Delta|$ is also reduced. Therefore, to let the weight assigner g be sensitive about the difference of shots (reflected by the uncertainty) between the base and novel classes, we set g_1 as $\texttt{MLP}(\mathbf{X})$ to retain the uncertainty gap as large as possible.

We also provide the visualization of prediction vectors for the base and novel classes in Fig. 3.12. Remark that the prediction vectors are collected from 10 runs and in every run, the base/novel split is different. Naturally, if the model's uncertainty gap between the base and novel classes is larger, the prediction vectors of the base and novel nodes should be more differentiable. Clearly, the visualization results align well with our conclusion from the above case study.

CHAPTER 4: GRAPH AUGMENTATION

4.1 NODE CLASSIFICATION BEYOND HOMOPHILY: TOWARDS A GENERAL SOLUTION

4.1.1 Introduction

Graph neural networks (GNNs) have demonstrated great power as building blocks for a variety of graph learning tasks, such as node classification [20, 159], graph classification [160], link prediction [161, 162], clustering [163], and many more. Most of the existing GNNs follow the homophily assumption, i.e., edges tend to connect nodes with the same labels and similar node features. Such an assumption holds for networks such as citation networks [137, 164] where a paper tends to cite related literature. However, in many other cases, the *heterophilic* settings arise. For instance, to form a protein structure, different types of amino acids are more likely to be linked together [32]. On such heterophilic networks, the performance of classic GNN models [21, 23, 165] could degrade significantly and might be even worse than a multilayer perceptron (MLP) which does not utilize any topology information at all [32].

In response, researchers have analyzed the limitations of existing GNNs in the presence of node heterophily and proposed specific models to address this issue from both spatial and spectral perspectives. For instance, an important design by H2GCN [32] is that high-order neighbors should be considered during message aggregation. GPRGNN [166] also aggregates messages from multi-hop neighbors but it emphasizes that messages can also be negative via a set of learnable aggregation weights. From the spectral perspective, FAGCN [167] points out that low-pass filter-based GNNs smooth the node representations between connected nodes, which is not desirable for the heterophilic settings where connected nodes are more likely to have different labels. Hence, FAGCN [167] adaptively mixes the low-pass graph filter with the high-pass graph filter via an attention mechanism to tackle this problem.

Despite the theoretical insights and empirical performance gains, most existing works focus on the model level, i.e., they aim to propose better GNN models to handle the heterophilic graphs. In other words, the success of their methods relies on specific designs of GNN models. In this paper, we take a step further and ask: *how to develop a generic method to benefit a broad range of GNNs for node classification beyond homophily, even if they are not tailored initially for the heterophilic graphs?* To this end, we address this problem from a structure learning [168] perspective, that is, we optimize the given graph structure to benefit downstream tasks (e.g., node classification). Unlike existing approaches that refine specific

GNN models, our approach focuses on the data level by optimizing the graph topology to address heterophily.

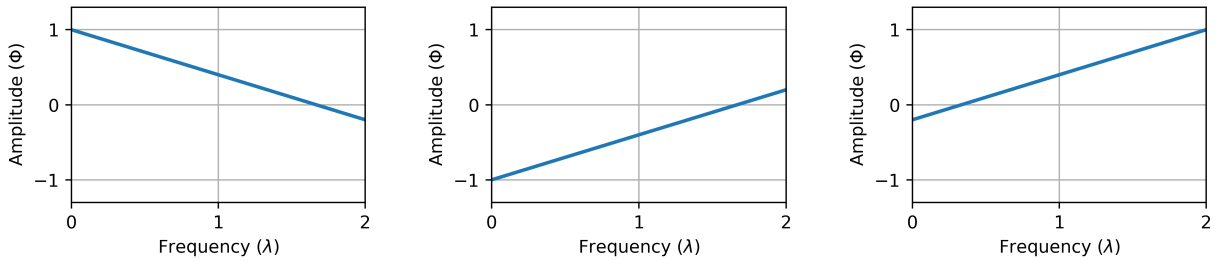
Challenges. In pursuing a data-centric general solution, the following are the key challenges. First (*model diversity*), our goal is to enhance the capabilities of a broad range of established GNNs, enabling them to handle graphs with arbitrary homophily. However, the aggregation mechanism and graph convolution kernels differ between various GNN models. It is unknown how to accommodate diverse GNNs seamlessly. Second (*theoretical foundation*), analyses on the success of some specific GNNs for heterophilic graphs have recently emerged (e.g., from the graph signal processing perspective [169]). However, few works focus on the theoretical foundation of structure learning and its connection to dealing with graphs with low homophily.

4.1.2 Preliminaries

Notations. We use bold uppercase letters for matrices (e.g., $\mathbf{A}$), bold lowercase letters for column vectors (e.g., $\mathbf{u}$), lowercase and uppercase letters in regular font for scalars (e.g., d, K), and calligraphic letters for sets (e.g., $\mathcal{T}$). We use $\mathbf{A}[i,j]$ to represent the entry of matrix $\mathbf{A}$ at the i-th row and the j-th column, $\mathbf{A}[i,:]$ to represent the i-th row of matrix $\mathbf{A}$, and $\mathbf{A}[:,j]$ to represent the j-th column of matrix $\mathbf{A}$. Similarly, $\mathbf{u}[i]$ denotes the i-th entry of vector $\mathbf{u}$. Superscript $\top$ denotes the transpose of matrices and vectors. $\odot$ denotes the Hadamard product.

An attributed graph can be represented as $\mathcal{G} = \{\mathbf{A}, \mathbf{X}\}$ which is composed of an adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ and an attribute matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, where n is the number of nodes and d is the node feature dimension. In total, nodes can be categorized into a set of classes C. The normalized Laplacian matrix is $\tilde{\mathbf{L}} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ where $\mathbf{D}$ is the diagonal degree matrix of $\mathbf{A}$. It can be decomposed as $\tilde{\mathbf{L}} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$ where $\mathbf{U} \in \mathbb{R}^{n \times n}$ is the eigenvector matrix and $\mathbf{\Lambda} \in \mathbb{R}^{n \times n}$ is the diagonal eigenvalue matrix. In graph signal processing [169], the diagonal entry of $\mathbf{\Lambda}$ represents frequency and $\mathbf{\Lambda}[i,i] = \lambda_i$. Given a signal $\mathbf{x} \in \mathbb{R}^n$, its graph Fourier transform [169] is represented as $\hat{\mathbf{x}} = \mathbf{U}\mathbf{x}$, and its inverse graph Fourier transform is defined as $\mathbf{x} = \mathbf{U}^\top\hat{\mathbf{x}}$. For a diffusion matrix $\mathbf{C} \in \mathbb{R}^{n \times n}$, its frequency response (or profile [170]) is defined as $\mathbf{\Phi}_{\mathtt{fp}} = \mathtt{diag}(\mathbf{U}^\top\mathbf{C}\mathbf{U})$ where $\mathtt{diag}(\cdot)$ returns the diagonal entries. This frequency response is also known as the filter and the convolution kernel.

Semi-supervised Node Classification. In this paper, we study *semi-supervised node classification* [20, 164] where the graph topology $\mathbf{A}$, all node features $\mathbf{X}$, and a part of node labels are given and our goal is to predict the labels of unlabelled nodes. Numerous works [20, 21, 23] achieve impressive performance on this problem. However, recent studies show that



(a) A given filter (b) Reflection (c) Reflection + offset

Figure 4.1: The Illustration of obtaining a filter with complementary filter characteristics. Given a filter (a), its reflected frequency response (b) with offset (c) has complementary filter characteristics.

their successes heavily rely upon the homophily assumption of the given graphs [32, 171]. In general, homophily describes to what extent edges tend to link nodes with the same labels and similar features. Following previous works [32, 172], this paper focuses on the node label homophily. There are various homophily metrics and we introduce one of them named edge homophily [32] as: $h(\mathcal{G}) = \frac{\sum_{i,j,\mathbf{A}[i,j]=1} [\![\mathbf{y}[i]=\mathbf{y}[j]]\!]}{\sum_{i,j} \mathbf{A}[i,j]} \in [0,1]$, where $[\![x]\!] = 1$ if x is true and 0 otherwise. The more homophilic a given graph is, the closer its $h(\mathcal{G})$ is to 1.

4.1.3 Proposed Methods

In this section, we first propose a flexible method named ALT-global, which empowers a wide range of GNNs with **adaptive filter** characteristics. Next, we carefully analyze the expressiveness of ALT-global from the graph signal processing perspective [169]. This analysis guides the design of another more advanced method named ALT-local, which enhances the spectral expressiveness of a broad range of GNNs to be local adaptive filters by **modulating** the input graph signals.

ALT-global: A Global Adaptive Method. Intuitively, nodes with different labels should be located as far as possible in the embedding space, and nodes with the same labels should be assigned closely. This intuition is aligned well with the utility of many classic GNNs (e.g., GCN [20]) on homophilic graphs. That is because, on homophilic graphs, many same-label nodes are connected, whose embeddings will be smoothed by those classic low-pass filter GNNs [167, 170]. In contrast, the performance of low-pass filter GNNs on heterophilic graphs degrades significantly, as the embeddings of connected nodes should not be smoothed. Many efforts [166, 167, 173] highlight that a key approach for handling graphs with unknown homophily is to equip GNNs with an adaptive filter.

We aim to propose a data-centric solution such that minimal modification on the given GNNs (e.g., a low-pass filter GNN) is needed. As we do not make any assumption about the filter characteristic of the given GNN, its filter can be either low-pass, high-pass, band-pass, or others. To equip the given GNN with an adaptive filter, our core idea is to *adaptively combine signals from two filters with the complementary filter characteristics*. For example, if a low-pass filter GNN is given, it should be adaptively combined with another high-pass filter. To find such a complementary filter, a two-step modification of the frequency response is needed. Figure 4.1 shows that we can first *reflect the frequency response curve over the frequency axis* and then *set an appropriate offset to the reflected frequency response*. Guided by this idea, the mathematical details of the proposed ALT-global are as follows,

$$\mathbf{H}_1 = \texttt{GNN}\left(w\mathbf{A}, \mathbf{X}, \theta_1\right), \tag{4.1a}$$

$$\mathbf{H}_2 = \texttt{GNN}\left((1-w)\,\mathbf{A}, \mathbf{X}, \theta_2\right), \tag{4.1b}$$

$$\mathbf{H}_{\texttt{offset}} = \texttt{MLP}\left(\mathbf{X}, \theta_3\right), \tag{4.1c}$$

$$\mathbf{Z} = \texttt{softmax}\left(\mathbf{H}_1 - \mathbf{H}_2 + \eta\mathbf{H}_{\texttt{offset}}\right), \tag{4.1d}$$

where θ_1 and θ_2 are the parameters of the **backbone dual GNNs** (i.e., GNNs from Eq. 4.1a and Eq. 4.1b), θ_3 is the parameter of a multi-layer perceptron (MLP), $\eta \in \mathbb{R}$ and $w \in [0, 1]$ are learnable parameters, and $\mathbf{Z} \in \mathbb{R}^{n\times C}$ is the prediction matrix. Here the `softmax` is applied row-wise. For models using the normalized adjacency matrix (e.g., $\tilde{\mathbf{A}} = (\mathbf{D}+\mathbf{I})^{-\frac{1}{2}}(\mathbf{A}+\mathbf{I})(\mathbf{D}+\mathbf{I})^{-\frac{1}{2}}$) as the diffusion matrix (e.g., GCN [20]), the re-weighting can be set over the normalized adjacency matrix (i.e., $w\tilde{\mathbf{A}}$ and $(1-w)\tilde{\mathbf{A}}$).

We elaborate more on the design of ALT-global. First, all the insights we obtained from Figure 4.1 apply to the convolution kernel directly. Nonetheless, since our method works in a plug-and-play fashion that does not modify the backbone GNNs, it uses a well-designed aggregation (i.e., Eq. 4.1d) to achieve an equivalent effect. Specifically, (1) $\mathbf{H}_1$ is the signals from a backbone GNN with positive re-scaling; (2) $-\mathbf{H}_2$ is the negative signals that correspond to the signals from a reflected filter; (3) $\eta\mathbf{H}_{\texttt{offset}}$ is the offset term, which is equivalent to signals from an all-pass filter. Second, the adaptive mixture of the above three sets of graph signals is controlled by the learnable parameters w and η. Other aggregation functions are also applicable. One of the options is an MLP whose input is the concatenation of $\mathbf{H}_1$, $\mathbf{H}_2$, and $\mathbf{H}_{\texttt{offset}}$. However, it is not used in this paper because (1) it increases the analysis difficulties dramatically and (2) empirically, no performance advantage is observed in the ablation study (Section 4.1.4). Analysis in the following section shows that ALT-global bears strong flexibility in filter characteristics.

Analysis of ALT-global. For brevity, in the analysis, we assume that the backbone GNNs are graph-augmented MLPs (GA-MLPs) as defined below. This is because many GNNs fall into the GA-MLP family if part of the nonlinear functions is removed; also, GA-MLPs have shown strong empirical performance while enjoying provable expressiveness [174].

Definition 4.1. Graph-Augmented Multi-Layer Perceptron (GA-MLP) [174] is a family of GNNs that first conduct feature transformation via an MLP and then diffuse the features. Mathematically they compute node embeddings as $\mathbf{H} = \mathbf{C} \cdot \texttt{MLP}(\mathbf{X})$ where $\mathbf{C}$ is the diffusion matrix.

The (full) frequency profile [170] is used for analysis as follows.

Definition 4.2. Frequency profile [170] is defined as
$\mathbf{\Phi}_{\texttt{fp}} = \texttt{diag}(\mathbf{U}^\top \mathbf{C} \mathbf{U})$ where $\texttt{diag}(\cdot)$ returns the diagonal entries if $\mathbf{U}^\top \mathbf{C} \mathbf{U}$ is a diagonal matrix. In case $\mathbf{U}^\top \mathbf{C} \mathbf{U}$ is not a diagonal matrix, **full frequency profile** [170] is defined as $\mathbf{\Phi} = \mathbf{U}^\top \mathbf{C} \mathbf{U}$.

It is well-known that the frequency profile of a diffusion matrix (if diagonal) is a filter/-convolution kernel for the input graph signal. Next, we show that ALT is indeed equipped with an adaptive filter.

Lemma 4.1. The filter characteristic of the proposed ALT-global (Eq. 4.1d) is adaptive regardless of the frequency filtering functionality of the backbone GNNs (Eq. 4.1a and Eq. 4.1b).

Proof. For analysis convenience, we assume (1) the learnable weight w is multiplied with the diffusion matrix $\mathbf{C}$, and (2) the backbone GNNs are GA-MLPs whose MLP modules (from Eq. 4.1a and Eq. 4.1b) share common parameters with the offset MLP (from Eq. 4.1c). We start from the case where backbone GNNs are fixed low-pass filters. Without loss of generality, their corresponding full frequency profiles can be presented as $\mathbf{\Phi} = \mathbf{I} - \xi(\mathbf{\Lambda})$ where ξ is a monotonically increasing function. Then, in this case, the diffusion matrices from two GNNs are re-weighted as $w\mathbf{C}$ and $(1-w)\mathbf{C}$ respectively. Considering the offset MLP as a special GA-MLP whose diffusion matrix is $\mathbf{I}$, the aggregated graph signals are $w\mathbf{C} \cdot \texttt{MLP}(\mathbf{X}) - (1-w)\mathbf{C} \cdot \texttt{MLP}(\mathbf{X}) + \eta \mathbf{I} \cdot \texttt{MLP}(\mathbf{X}) = \tilde{\mathbf{C}} \cdot \texttt{MLP}(\mathbf{X})$ where the aggregated diffusion matrix is $\tilde{\mathbf{C}} = w\mathbf{C} - (1-w)\mathbf{C} + \eta\mathbf{I}$. Hence, the diagonal entry of the corresponding full frequency profile is

$$\mathbf{\Phi}[i,i] = \Phi(\lambda_i) = (2w-1)(1-\xi(\lambda_i)) + \eta. \tag{4.2}$$

When $w > 0.5$, i.e., $2w - 1 > 0$, $\Phi(\lambda_i)$ is a monotonically decreasing function. The proposed method is a low-pass filter when $\eta > 0$. Similarly, it is a high-pass filter when w is close

to 0 and $\eta > 1$. The above conditions are sufficient, and in fact, there are many other combinations of w and η which lead to low-pass/high-pass filters. Similar results can be obtained when the backbone GNNs are fixed high-pass filters, and we omit that part for brevity. QED.

Remarks. The filter characteristics of the ALT-global can also be interpreted from the Graph Diffusion Equation (GDE) [175] perspective, and we provide the GDE-related analysis in the following content.

Global Filters vs. Local Filters. ALT-global is proved to be equipped with adaptive filter characteristics. However, ALT-global fundamentally applies a global filter to every node, which could lead to suboptimal performance. Recent studies [176, 177] reveal that heterophilic connection patterns differ between different nodes. Take gender classification on a dating network as an example. While node pairs are often of different labels (i.e., genders), homosexuality also exists between some node pairs. Therefore, simply applying a global low-pass or high-pass filter over all the nodes can degrade the overall classification performance.

Next, we study how to generalize our proposed ALT-global to a local (i.e., node-specific) and adaptive filter. Before that, let us take a closer look at the full frequency profile [170]: $\mathbf{\Phi} = \mathbf{U}^\top \mathbf{C} \mathbf{U}$. In the following proposition, we point out that $\mathbf{\Phi}$ can describe both the filter and modulator characteristics of a given diffusion matrix $\mathbf{C}$.

Proposition 4.1. The diagonal entries of the full frequency profile $\mathbf{\Phi}$ of the diffusion matrix serve as the **filter** and the non-zero off-diagonal entries are the **frequency modulator**.

Proof. The diffusion of the input graph signal $\mathbf{X}_{\texttt{in}} = \texttt{MLP}(\mathbf{X})$ can be represented as $\mathbf{C}\mathbf{X}_{\texttt{in}} = \mathbf{U}\mathbf{\Phi}\mathbf{U}^\top \mathbf{X}_{\texttt{in}} = \mathbf{U}(\mathbf{\Phi}\hat{\mathbf{X}}_{\texttt{in}})$, where $\hat{\mathbf{X}}_{\texttt{in}}$ is the input graph signal in spectral domain. From the perspective of graph signal processing [169], $(\mathbf{\Phi}\hat{\mathbf{X}}_{\texttt{in}})[i:]$ represents the amplitude of output graph signal whose frequency is λ_i. We further expand the computation and obtain

$$(\mathbf{\Phi}\hat{\mathbf{X}}_{\texttt{in}})[i:] = \sum_j \mathbf{\Phi}[i,j] \cdot \mathbf{X}_{in}[j,:]. \tag{4.3}$$

In the summation, the diagonal terms of $\mathbf{\Phi}$ represent the filter/convolution kernel which has been adopted by many spectral GNNs [170]. If non-zero off-diagonal entries of $\mathbf{\Phi}$ exist, it shows that the λ_i-component of the output graph signal is merged with scaled (by $\mathbf{\Phi}[i,j]$) λ_j-component of the input graph signal which is essentially the modulation [169]. QED.

Based on the above property of the full frequency profile $\mathbf{\Phi}$, the following proposition points out the key design for local filter characteristics.

Proposition 4.2. Modulation of the input graph signal (i.e., **non-zero off-diagonal entries** of $\mathbf{\Phi}$) is necessary for **local** filters.

Proof. We follow the terminology used in the proof of Proposition 4.1. If the full frequency profile $\mathbf{\Phi}$ only contains non-zero diagonal entries, we can obtain

$$(\mathbf{\Phi}\hat{\mathbf{X}}_{\texttt{in}})[i,:] = (\texttt{diag}(\mathbf{\Phi}))^{\top} \odot \hat{\mathbf{X}}_{\texttt{in}}[i,:], \tag{4.4}$$

where $\texttt{diag}$ extracts the diagonal entries into a vector from the input square matrix. Hence, if we define the scaling of the λ_i-frequency signal over node p after and before the operator $\mathbf{\Phi}$ as $\texttt{SCALING}(i,p,\mathbf{\Phi}) = \frac{(\mathbf{\Phi}\hat{\mathbf{X}}_{\texttt{in}})[i,p]}{\hat{\mathbf{X}}_{\texttt{in}}[i,p]}$, from Eq. (4.4) we have

$$\forall i,p,q, \quad \texttt{SCALING}(i,p,\mathbf{\Phi}) = \texttt{SCALING}(i,q,\mathbf{\Phi}) \tag{4.5}$$

i.e., for any specific frequency (e.g., λ_i), its scaling over any two nodes (p and q) are equal. In other words, the filter $\mathbf{\Phi}$ works globally over every node. If we expect the filter $\mathbf{\Phi}$ to not work globally, i.e.

$$\exists i,p,q, \quad \texttt{SCALING}(i,p,\mathbf{\Phi}) \neq \texttt{SCALING}(i,q,\mathbf{\Phi}). \tag{4.6}$$

The above inequality is equivalent to

$$\frac{\sum_{k,k\neq i} \mathbf{\Phi}[i,k] \cdot \hat{\mathbf{X}}_{\texttt{in}}[k,p]}{\hat{\mathbf{X}}_{\texttt{in}}[i,p]} \neq \frac{\sum_{k,k\neq i} \mathbf{\Phi}[i,k] \cdot \hat{\mathbf{X}}_{\texttt{in}}[k,q]}{\hat{\mathbf{X}}_{\texttt{in}}[i,q]}. \tag{4.7}$$

Assume that $\forall k$, if $k \neq i$, $\mathbf{\Phi}[i,k] = 0$, and then the left-hand side is equal to the right-hand side which leads to a contradiction. Hence, non-zero off-diagonal entries of the full frequency profile $\mathbf{\Phi}$ must exist if we expect the filter to not work globally. Notice that the above definition of scaling (e.g., $\frac{(\mathbf{\Phi}\hat{\mathbf{X}}_{\texttt{in}})[i,p]}{\hat{\mathbf{X}}_{\texttt{in}}[i,p]}$) is not fully aligned with the classic graph filtering [169] but a combination of filtering and modulation as we mentioned in Proposition 4.1. QED.

Next, we present a family of GA-MLPs whose spectral expressiveness is limited to a global filter.

Proposition 4.3. A family of GA-MLPs are global filters if their full frequency profiles are in the form of $\mathbf{C} = \sum_k a_k \tilde{\mathbf{A}}^k + b\mathbf{I}$ which only contains non-zero diagonal entries.

Proof. Since $\{\tilde{\mathbf{A}}^k\}$ and $\mathbf{I}$ share the same eigenvectors, the diffusion matrix can be decomposed

as

$$\mathbf{C} = \sum_k a_k \tilde{\mathbf{A}}^k + b\mathbf{I} = \mathbf{U}\left(\sum_k a_k \left(\mathbf{I} - \tilde{\mathbf{\Lambda}}\right)^k + b\mathbf{I}\right)\mathbf{U}^\top. \tag{4.8}$$

Hence, the frequency profile is $\mathbf{\Phi} = \sum_k a_k \left(\mathbf{I} - \tilde{\mathbf{\Lambda}}\right)^k + b\mathbf{I}$ whose off-diagonal entries are zero. QED.

A wide range of GA-MLPs (e.g., SGC [22], APPNP [23]) follow the above form and therefore cannot modulate graph signal. Unfortunately, even when they are equipped with our proposed ALT-global, they are still *global* filters because ALT-global assigns the same weight to every edge (i.e., $w\tilde{\mathbf{A}}$ and $(1-w)\tilde{\mathbf{A}}$).

ALT-local: A Local Adaptive Method. In this subsection, we propose a more flexible method based on ALT-global. Our goal is to empower the backbone GNNs with local adaptive signal filtering capabilities, which is an essential property for capturing complex heterophilic connection patterns [176, 177]. According to Proposition 4.3, we know that if all the edges are assigned with the same weight (e.g., $w\tilde{\mathbf{A}}$) the corresponding full frequency profile will only contain diagonal non-zero entries. Lemma 4.2 provides a critical clue on how to bring non-zero off-diagonal entries in full frequency profiles.

Lemma 4.2. By re-weighting the edge weights non-uniformly (i.e., if re-weighting by $\mathbf{W} \odot \tilde{\mathbf{A}}$, $\exists i, j, k, l, \mathbf{W}[i,j] \neq \mathbf{W}[k,l]$), the off-diagonal entries of $\mathbf{\Phi}$ can be non-zero.

Proof. We follow the assumption mentioned in the proof of Lemma 4.1. The diffusion matrix $\mathbf{C}$ can be decomposed as $\mathbf{C} = \mathbf{U}\mathbf{\Phi}\mathbf{U}^\top$. For the full frequency profile $\mathbf{\Phi}$, its off-diagonal entry $\mathbf{\Phi}[i,j] = \sum_{l,k} \mathbf{U}[l,i]\mathbf{C}[l,k]\mathbf{U}[k,j] = 0, \forall i \neq j$. If we re-weight the diffusion matrix by $\mathbf{W} \odot \mathbf{C}$ such that $\mathbf{W}[l,k] = w_{lk}$ and $\mathbf{W}[i,j] = w \neq w_{lk}, \forall i \neq l \; and \; j \neq k$. In other words, we start from the most basic case where only one edge (l,k) ($\mathbf{C}[l,k] \neq 0$) is re-weighted by w_{lk} and all the remaining edges are re-weighted as w. Given the zero off-diagonal entries of $\mathbf{\Phi}$ we have

$$\begin{aligned} \mathbf{\Phi}_{\texttt{re-weighted}}[i,j] &= \left(\mathbf{U}^\top(\mathbf{W} \odot \mathbf{C})\mathbf{U}\right)[i,j] \\ &= \left(\mathbf{U}^\top(\mathbf{W} \odot \mathbf{C})\mathbf{U}\right)[i,j] - w\mathbf{\Phi}[i,j] \\ &= \mathbf{U}[l,i]\mathbf{C}[l,k]\mathbf{U}[k,j](w_{lk} - w). \end{aligned} \tag{4.9}$$

It is common to find i and j ($j \neq i$) such that $\mathbf{U}[l,i]\mathbf{U}[k,j] \neq 0$, and thus we have $\mathbf{\Phi}_{\texttt{re-weighted}}[i,j] \neq 0$ as long as $w_{lk} \neq w$. Therefore, we proved that if the edge weights are re-weighted non-uniformly, the off-diagonal entries of $\mathbf{\Phi}$ can be non-zero, i.e., the GNN can be a local filter. QED.

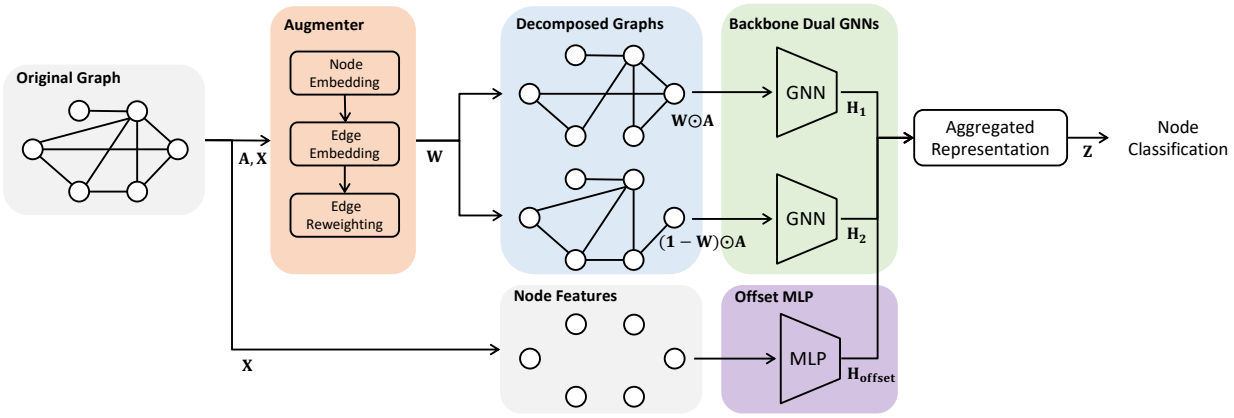


Figure 4.2: The proposed ALT-local.

Guided by Lemma 4.2 we modify ALT-global as follows so that the edge weights are different:

$$\mathbf{H}_1 = \texttt{GNN}(\mathbf{W} \odot \mathbf{A}, \mathbf{X}, \theta_1), \tag{4.10a}$$

$$\mathbf{H}_2 = \texttt{GNN}((\mathbf{1} - \mathbf{W}) \odot \mathbf{A}, \mathbf{X}, \theta_2), \tag{4.10b}$$

$$\mathbf{H}_{\texttt{offset}} = \texttt{MLP}\left(\mathbf{X}, \theta_3\right), \tag{4.10c}$$

$$\mathbf{Z} = \texttt{softmax}\left(\mathbf{H}_1 - \mathbf{H}_2 + \eta \mathbf{H}_{\texttt{offset}}\right), \tag{4.10d}$$

One option is to set $\mathbf{W}$ as a learnable parameter which is prune to overfitting as the number of parameters is equal to the number of edges. Therefore, we parameterize the edge weight $\mathbf{W}$ by an edge augmenter as follows,

$$\mathbf{H} = \texttt{GNN}_{\texttt{aug}}\left(\mathbf{A}, \mathbf{X}, \phi_1\right), \tag{4.11a}$$

$$\mathbf{W}[i,j] = w_{ij} = \texttt{sigmoid}(\texttt{MLP}\left(\mathbf{H}[i,:]||\mathbf{H}[j,:], \phi_2\right)) \tag{4.11b}$$

where ϕ_1 and ϕ_2 are the parameters of the augmenter GNN and a multi-layer perceptron (MLP) respectively. Here we first obtain the node embedding matrix via the augmenter GNN (i.e., $\texttt{GNN}_{\texttt{aug}}$) in Eq. 4.11a. Then we concatenate node embeddings into edge embeddings (i.e., $\mathbf{H}[i,:]||\mathbf{H}[j,:]$). The edge weight (i.e., w_{ij}) is computed via an MLP with `sigmoid` activation. Naturally, the node embeddings from the augmenter GNN (Eq. 4.11a) should be as discriminative as possible so that the edge importance can be better measured. Thus, we use a two-layer high-pass filter GNN as the $\texttt{GNN}_{\texttt{aug}}$ whose mathematical formulation is as

follows,

$$\texttt{GNN}_{\texttt{aug}}(\mathbf{A}, \mathbf{X}, \phi_1) = \tilde{\mathbf{A}}^2_{\texttt{high}}\texttt{MLP}(\mathbf{X}, \phi_1), \tag{4.12a}$$

$$\tilde{\mathbf{A}}_{\texttt{high}} = \epsilon\mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}, \tag{4.12b}$$

where ϵ is a scaling hyperparameter to adjust the amplitude of the high-pass filter. We name the above model (i.e., Eqs.4.10a-4.12b) as ALT-local , which is summarized in Figure 4.2.

Remarks. Our method is partly inspired by FAGCN [167]. We clarify the uniqueness and advantages of our work in comparison to FAGCN as follows. From a methodological perspective, FAGCN explicitly combines high-frequency and low-frequency signals. ALT generalizes this idea to the 'mixture of complementary filters'; thus, even though the backbone GNN's convolution kernel is unknown, ALT can still boost its performance decently, which provides great generality. For the theoretical analysis, [167] analyzes the spatial effects of signals with different frequencies. Our analysis takes a solid step forward to reveal the intrinsic connections between (i) the full frequency profile, (ii) graph signal modulation, and (iii) local adaptive filters.

Training Objective. The optimization objective of ALT is as follows.

$$\phi^*, \theta^* = \arg\min_{\theta,\phi} \mathcal{L}_{\texttt{cla}}(g(\mathcal{G}, \phi), \theta, \mathcal{Y}) \tag{4.13}$$

where the augmenter is denoted as $g(\cdot)$ whose parameter is ϕ and the dual backbone GNNs are parameterized as θ for brevity. Specifically, for the ALT-global, $\theta = \{\theta_1, \theta_2, \theta_3\}$ and $\phi = w$ are from Eq.4.1a, Eq.4.1b, and Eq.4.1c. For ALT-local, $\theta = \{\theta_1, \theta_2, \theta_3\}$ is from Eq. 4.10a, 4.10b, and Eq. 4.10c; $\phi = \{\phi_1, \phi_2\}$ is from Eq. 4.11a and 4.11b. $\mathcal{L}_{cla}$ is cross-entropy loss between the classification results (Eq. 4.1d for ALT-global and Eq. 4.10d for ALT-local) and the labeled nodes.

If all the feature dimensions of different layers (including the input layers) from different backbone GNNs and MLPs are denoted as d and all the models (GNNs and MLPs) contain 2 feature transformation matrices, the number of trainable parameters of ALT-local is composed of three parts: (1) $\texttt{GNN}_{\texttt{aug}}$ ($2d^2$), (2) MLP from Eq. 4.11b ($2d^2 + d$), (3) $\texttt{GNN}_1$, $\texttt{GNN}_2$, and offset MLP ($3d^2 + 3dc$) where c is the number of classes. In practice, the parameter number is much smaller than the estimated number. For example, for datasets whose $d > 500$, empirically, setting the hidden dimension as 32 is enough. However, compared with vanilla backbone GNNs (e.g., a simple GCN [20]), ALT-local inevitably contains more parameters as ALT-local is composed of 3 GNNs and 2 MLPs in total. Even for ALT-global, it is still

Table 4.1: Performance comparison (mean±std accuracy) on heterophilic graphs. The last column indicates the average performance boost for a specific backbone GNN for all the datasets.

Backbone	ALT?	Chameleon	Squirrel	Texas	Wisconsin	Cornell	Film	Cornell5	Penn94	Avg. Δ
GCN	No	58.4±1.1	35.4±0.6	57.6±3.5	51.2±1.6	55.9±1.6	28.1±0.3	72.1±0.1	74.7±0.3	+12.4
	Yes	65.8±0.9	52.4±0.8	70.9±4.3	76.4±3.9	73.9±5.1	35.5±1.2	77.5±0.1	80.1±0.4	
SGC	No	58.4±0.6	37.1±0.4	58.6±1.9	48.3±1.8	57.0±3.4	27.3±0.1	72.6±0.4	75.0±0.4	+11.9
	Yes	65.6±2.0	53.2±0.6	71.5±2.8	72.8±1.6	72.1±9.0	34.9±0.8	79.0±0.3	80.2±0.1	
APPNP	No	48.0±1.2	33.8±0.4	59.5±1.1	48.8±2.0	56.3±1.4	28.7±0.3	70.6±0.5	73.4±0.4	+15.1
	Yes	65.4±1.1	53.2±0.9	71.2±2.9	76.6±2.7	78.4±3.4	34.0±0.3	79.7±0.1	81.7±0.4	
GPRGNN	No	59.2±2.5	38.4±0.8	69.1±1.0	72.4±1.6	69.6±2.5	32.1±1.1	74.3±1.3	78.7±0.3	+7.1
	Yes	66.7±0.9	53.0±1.0	75.4±2.4	79.7±0.5	70.6±1.5	32.8±1.0	80.2±0.8	82.3±0.4	
FAGCN	No	54.3±1.9	32.5±1.4	61.5±1.3	56.6±5.2	66.0±1.7	33.8±0.7	69.1±0.2	72.8±0.3	+11.1
	Yes	64.5±1.0	52.8±1.4	69.4±0.7	76.4±5.7	75.1±6.8	35.7±0.5	79.9±0.1	81.9±0.4	
H2GCN	No	49.9±1.4	31.5±0.8	67.6±2.1	70.4±2.1	69.4±3.3	34.5±0.3	69.5±0.4	73.5±0.1	+8.1
	Yes	61.5±0.7	51.6±0.5	76.0±4.7	77.7±4.4	78.4±3.4	35.7±0.3	71.4±0.2	78.7±0.8	

composed of 2 GNNs and 1 MLP. Hence, the increased number of parameters is a potential limitation of ALT-local and ALT-global.

4.1.4 Experiments

Datasets. 16 datasets are used including Cora [164], Citeseer [164], Pubmed [164], DBLP [137], Computers [178], Photos [178], CS [178], Physics [178], Cornell [172], Texas [172], Wisconsin [172], Chameleon [179], Squirrel [179], Film [172], Cornell5 [180], and Penn94 [180]. We obtain all the datasets from pytorch-geometric[1], which are publicly available. In the effectiveness study, in order to compare with the state-of-the-art methods, we adopt the dataset split 48/32/20% (training/validation/test) from a recent work ACM-GCN [181]. In the other subsections, to fully test the applicability of ALT, we use the following challenging dataset split: (1) we follow the given dataset split for Cora (8.5/30.5/61.0%), Citeseer (7.4/30.9/61.7%), and Pubmed (3.8/32.1/64.1%); (2) for the remaining datasets, we randomly split them into 20/20/60% (training/validation/test). Detailed statistics of the datasets are presented in Table 4.2 and Table 4.3.

Accuracy (ACC) is adopted as the metric. We report the average accuracy with the standard deviation in 10 runs. The code is available[2].

Applicability of ALT. As the primary goal of this paper is to propose a general solution for handling graphs with arbitrary homophily, this section examines the applicability

[1]https://pytorch-geometric.readthedocs.io/en/latest/modules/datasets.html
[2]https://github.com/pricexu/ALT

Table 4.2: Dataset statistics of heterophilic graphs.

Dataset	**Chameleon**	**Squirrel**	**Texas**	**Wisconsin**	**Cornell**	**Film**	**Cornell5**	**Penn94**
# Nodes	2,277	5,201	183	251	183	7,600	18,660	41,554
# Edges	62,792	396,846	325	515	298	30,019	1,581,554	2,724,458
# Features	2,325	2,089	1,703	1,703	1,703	932	4,735	4,814
# Classes	5	5	5	5	5	5	2	2
$h(\mathcal{G})$	0.231	0.222	0.108	0.196	0.305	0.219	0.479	0.470

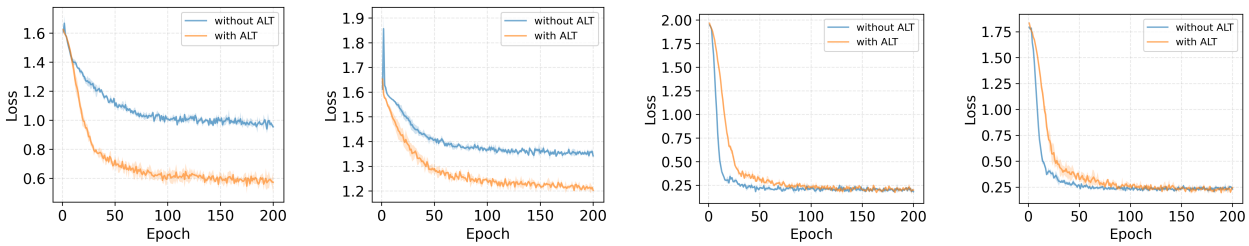
Table 4.3: Dataset Statistics of homophilic graphs.

Dataset	**Cora**	**Citeseer**	**Pubmed**	**DBLP**	**Computers**	**Photos**	**CS**	**Physics**
# Nodes	2,708	3,327	19,717	17,716	13,752	7,650	18,333	34,493
# Edges	10,556	9,104	88,648	105,734	491,722	238,162	163,788	495,924
# Features	1,433	3,703	500	1,639	767	745	6,805	8,415
# Classes	7	6	3	4	10	8	15	5
$h(\mathcal{G})$	0.810	0.736	0.802	0.828	0.777	0.827	0.808	0.931

of the proposed approach, ALT. Specifically, we select 6 representative backbone node classifiers including 3 classic GNNs: GCN [20], SGC [22], and APPNP [23], and 3 adaptive GNNs: GPRGNN [166], FAGCN [167], and H2GCN [32] which use specific designs to tackle graphs with low homophily. We aim to compare the performance improvement of the above backbone classifiers after being equipped with ALT. As ALT-local is more powerful than ALT-global, we mainly show the performance improvement after being equipped with ALT-local (short as ALT). The comparison between ALT-local and ALT-global will be presented in the ablation study below.

We present the performance comparison on heterophilic graphs in Table 4.1 and have the following observations. First, on the heterophilic graphs, in general, our method ALT can significantly improve the performance of most of the existing GNNs, especially for methods originally not designed for the heterophilic graphs (e.g., GCN, SGC, and APPNP), whose performance, on average, is improved by over 10%. Second, over the heterophilic graphs, the performance improvement of adaptive GNNs (e.g., GPRGNN, FAGCN, and H2GCN) is not as significant as that of low-pass filter GNNs. This is expected, as these methods have already addressed heterophily to some extent. Nonetheless, we still gain 7-11% performance improvements averaged over all 8 heterophilic datasets.

The performance comparison on homophilic graphs is presented in Table 4.4. We test 48 graph-GNN combinations, out of which, 29 cases show accuracy improvements $\geq 0.5\%$. It is worth noting that even though GCN, SGC, and APPNP are designed mainly for homophilic graphs, the proposed ALT is still able to boost their performance on Computers by nearly 18% significantly. Moreover, for each backbone GNN, the average gain of applying the



(a) Chameleon (b) Film (c) Cora (d) Citeseer

Figure 4.3: Training losses of APPNP (with/without ALT).

Table 4.4: Performance comparison (mean±std accuracy (%)) on homophilic graphs. The last column indicates the average performance boosting for a specific backbone GNN over all the datasets.

Backbone	ALT?	Cora	Citeseer	Pubmed	DBLP	Computers	Photos	CS	Physics	Avg. Δ
GCN	No	81.1±0.3	71.2±0.7	79.0±0.4	83.7±0.1	66.2±1.0	84.1±0.5	88.2±0.2	95.3±0.1	+3.4
	Yes	81.2±0.5	71.4±0.4	79.1±0.9	83.7±0.4	84.1±0.1	88.9±0.6	92.3±0.4	95.6±0.9	
SGC	No	80.8±0.1	71.0±0.2	79.5±0.5	83.8±0.0	69.1±0.4	86.2±0.4	89.7±0.1	95.3±0.0	+2.6
	Yes	80.7±0.4	71.2±0.6	79.5±0.7	83.7±0.0	84.0±0.4	88.8±1.4	92.5±0.3	96.0±0.0	
APPNP	No	82.1±0.1	71.8±0.1	79.8±0.5	83.8±0.2	66.7±1.1	83.4±1.2	87.8±0.1	94.9±0.0	+4.0
	Yes	82.7±0.3	72.1±0.3	79.3±0.2	84.6±0.1	84.6±0.4	88.7±0.3	93.8±0.1	96.4±0.1	
GPRGNN	No	78.6±1.5	68.9±0.9	77.6±0.9	84.4±0.2	85.0±0.5	92.4±0.2	92.3±0.1	95.5±0.4	+1.6
	Yes	83.0±0.4	71.0±0.4	80.3±0.2	85.1±0.2	85.8±0.2	92.9±0.2	93.4±0.2	96.2±0.1	
FAGCN	No	79.0±0.6	72.1±0.5	78.0±1.1	81.1±1.1	74.8±3.4	91.2±0.3	93.0±1.4	95.7±0.3	+1.8
	Yes	79.0±0.5	71.7±0.5	78.3±1.2	82.5±0.3	86.0±0.8	91.5±0.4	93.6±1.1	96.3±0.2	
H2GCN	No	78.9±0.6	70.3±1.0	78.2±1.0	82.4±0.0	75.8±0.3	89.7±0.2	92.5±0.2	96.2±0.1	+2.0
	Yes	79.0±0.4	70.9±0.8	78.0±1.3	82.0±0.4	87.0±0.3	92.0±0.6	94.1±0.2	96.6±0.1	

Table 4.5: Performance comparison (mean±std accuracy (%)) with the state-of-the-art methods. The best and the second best are bold and underlined, respectively. Results marked "*" are reported from [181] with the same dataset split.

Dataset	*ACM-GCN	BernNet	*LINKX	*ACMII-GCN++	*GloGNN++	ALT	ALT+
Cornell	85.1±6.1	81.1±8.4	77.8±5.8	86.5±6.7	86.0±5.1	86.8±4.3	**90.4±4.5**
Wisconsin	88.4±3.2	87.3±4.6	75.5±5.7	88.4±3.7	88.0±3.2	**88.9±2.5**	88.6±3.3
Texas	87.8±4.4	82.6±4.9	74.6±8.4	88.4±3.4	84.1±4.9	88.7±3.3	**89.5±2.2**
Film	36.6±0.8	34.2±1.5	36.1±1.6	37.1±1.3	**37.7±1.4**	37.6±0.7	37.3±1.2
Chameleon	69.1±1.9	45.4±1.9	68.4±1.4	74.8±2.2	71.2±1.8	66.7±2.0	**77.0±1.9**
Squirrel	55.2±1.5	33.1±1.4	61.8±1.8	67.4±2.2	57.9±1.8	54.3±1.2	**69.4±1.5**
Cora	87.9±1.0	87.6±0.6	84.6±1.1	88.3±1.0	88.3±1.1	88.1±0.5	**89.6±1.3**
Citeseer	77.3±1.7	76.1±0.3	73.2±1.0	77.1±1.6	77.2±1.8	77.6±1.5	**79.9±1.2**
PubMed	90.0±0.5	86.2±0.3	87.9±0.8	89.7±0.5	89.2±0.4	89.9±0.6	**90.3±0.5**

proposed ALT over all 8 homophilic graphs is always positive. Thus, we conclude that ALT can retain or even boost the performance of given backbone GNNs on homophilic graphs.

As we mentioned in Section 4.1.3, the model equipped with ALT will have more model parameters compared with a vanilla backbone GNN classifier. Thus, we further study the

Table 4.6: Ablation study with different backbone models.

(a) Backbone model: GCN

Backbone	Version	Chameleon	Squirrel	Film	Computers	Photos	CS
GCN	None	58.4±1.1	35.4±0.6	28.1±0.3	66.2±1.0	84.1±0.5	88.2±0.2
	Global	61.3±1.0	44.1±0.3	30.6±0.1	72.7±0.8	85.2±1.4	89.9±0.3
	Local-low	63.3±0.8	48.8±1.2	32.5±0.2	81.1±0.3	86.5±0.9	91.0±0.2
	Local-concat	47.1±2.6	31.3±1.4	34.4±1.1	76.4±5.8	85.3±3.7	87.1±1.2
	Local	**65.8±0.9**	**52.4±0.8**	**35.5±1.2**	**84.1±0.1**	**88.9±0.6**	**92.3±0.4**

(b) Backbone model: SGC

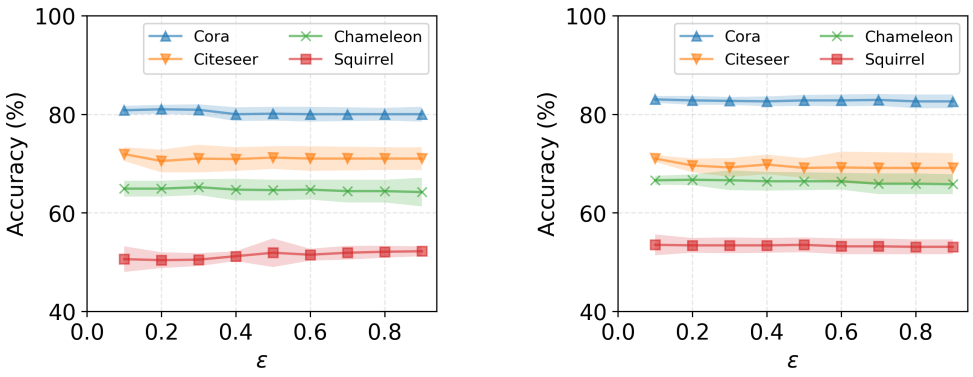
Backbone	Version	Chameleon	Squirrel	Film	Computers	Photos	CS
SGC	None	58.4±0.6	37.1±0.4	27.3±0.1	69.1±0.4	86.2±0.4	89.7±0.1
	Global	59.7±0.8	41.6±0.2	31.4±0.5	71.6±0.4	86.6±0.7	91.1±0.2
	Local-low	61.6±2.3	44.6±0.3	33.3±0.2	79.3±0.6	87.4±0.6	91.5±0.1
	Local-concat	44.0±5.9	36.4±1.7	34.0±1.9	79.6±2.1	88.1±3.0	90.2±0.7
	Local	**65.6±2.0**	**53.2±0.6**	**34.9±0.8**	**84.0±0.4**	**88.8±1.4**	**92.5±0.3**

(c) Backbone model: APPNP

Backbone	Version	Chameleon	Squirrel	Film	Computers	Photos	CS
APPNP	None	48.0±1.2	33.8±0.4	28.7±0.3	66.7±1.1	83.4±1.2	87.8±0.1
	Global	50.8±0.4	36.1±0.7	31.7±0.2	71.5±0.8	85.3±0.9	90.9±0.4
	Local-low	58.8±1.1	48.2±0.8	33.2±1.0	80.1±0.9	87.0±0.6	92.7±0.2
	Local-concat	51.9±0.7	40.0±1.0	33.6±0.7	75.5±2.3	81.8±2.5	89.9±0.4
	Local	**65.4±1.1**	**53.2±0.9**	**34.0±0.3**	**84.6±0.4**	**88.7±0.3**	**93.8±0.1**

training stability of a backbone classifier when working with ALT. To be specific, we select 2 homophilic datasets Cora and Citeseer, and 2 heterophilic datasets Chameleon and Film. We select the backbone classifier as APPNP. The training loss (negative log-likelihood loss) with respect to the number of epochs is reported in Figure 4.3a-4.3d from which we clearly observe that (1) the APPNP's training stability is not significantly affected after equipping ALT, (2) ALT-APPNP can fit the homophilic graphs as good as vanilla APPNP and, importantly, it can fit the heterophilic graphs much better (with lower training loss) than the vanilla APPNP. Observation (2) aligns well with our performance comparison reported in Table 4.1 and Table 4.4.

Effectiveness of ALT. In this section, we show that our proposed approach, ALT, can also be a strong competitor against state-of-the-art methods. We select APPNP [23] as our backbone method, which is not designed for graphs with high heterophily. Recent efforts to handle graphs with arbitrary heterophily are selected, which include LINKX [180], BernNet [182], ACM-GCN [181] and GloGNN [183]. For a fair comparison, we adopt the



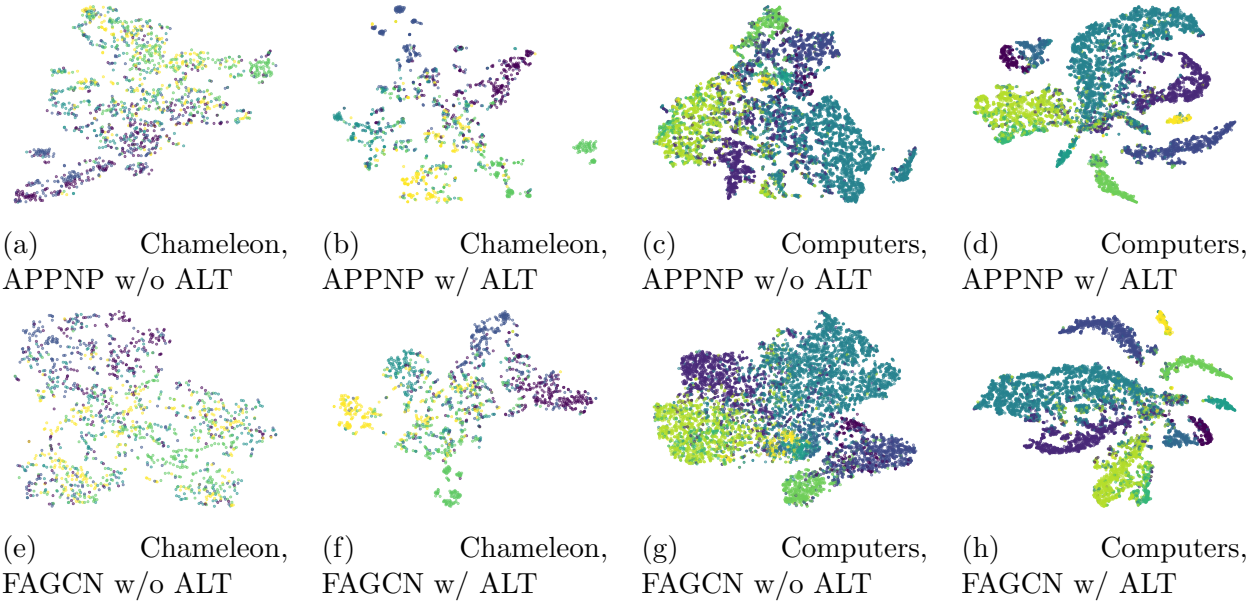
(a) GCN (b) GPRGNN

Figure 4.4: Hyperparameter sensitivity of ALT with backbone GNN as (a) GCN and (b) GPRGNN.

same dataset split as the recent work ACM-GCN [181]. The performance comparison is in Table 4.5. We observe that ALT-APPNP exhibits comparable performance to state-of-the-art methods on most datasets (except Chameleon and Squirrel). We notice that methods LINKX, ACM-GCN++, and GloGNN++ all use a technique to encode the adjacency matrix by an MLP (i.e., $\texttt{MLP}(\mathbf{A})$) as a supplement to node embeddings. Since this technique is independent of the model design, once it is applied to our framework, the model ALT-APPNP+ achieves very strong performance on the Chameleon and Squirrel datasets. In conclusion, our proposed ALT can be comparable to, or stronger than, state-of-the-art methods, even when working with a fixed-filter backbone GNN, such as APPNP.

Ablation Study. We present an ablation study on datasets: Chameleon [179], Squirrel [179], Film [172], Computers [178], Photos [178], and CS [178]. Specifically, we have the following ablated versions: (1) ALT-local, (2) ALT-local with a low-pass filter augmenter (i.e., change Eq.4.12b as a two-layer SGC) which is named as ALT-local-low, (3) ALT-local-concat whose aggregation step (Eq. 4.10d) is instantiated by 'concatenation' followed by an MLP, (4) ALT-global, and (5) vanilla backbone GNNs without our methods (named as None). Results are presented in Table 4.6, from which we observe that (1) the ALT-local has consistent advantages over all ablated versions, (2) the variant ALT-local-concat's performance is highly unstable which may be due to its large number of parameters in aggregating representations.

Hyperparameter Sensitivity Study. We study the sensitivity of ALT-local regarding the amplitude of the augmenter GNN (i.e, ϵ from Eq. 4.12b). We select GCN [20] and GPRGNN [166] as backbone GNNs and conduct experiments over Cora [164], Citeseer [164],



(a) Chameleon, APPNP w/o ALT (b) Chameleon, APPNP w/ ALT (c) Computers, APPNP w/o ALT (d) Computers, APPNP w/ ALT

(e) Chameleon, FAGCN w/o ALT (f) Chameleon, FAGCN w/ ALT (g) Computers, FAGCN w/o ALT (h) Computers, FAGCN w/ ALT

Figure 4.5: Visualization of backbone models with/without ALT on datasets Chameleon and Computers.

Chameleon [179], Squirrel [179] datasets. Results are presented in Figure 4.4 from which we observe that the model performance is stable for the selection of ϵ over four datasets and both backbone GNNs (i.e., GCN and GPRGNN).

Visualization. As a supplementary study of the model effectiveness, we visualize the node representations from the backbone models APPNP and FAGCN with/without our proposed ALT. To be specific, we use t-SNE [118] to map the representations of test nodes into two-dimensional vectors for visualization. We select a heterophilic graph Chameleon and a homophilic graph Computers. Figure 4.5a-4.5h show that after equipping with our proposed ALT (1) clusters of nodes with the same class (i.e., color in our visualization) are more cohesive in the embedding space and (2) backbone GNN's node representations from different classes are more discriminative.

Training Time Study. An analysis of the model complexity is provided at the end of subsection 5.2.3. Also, ALT does not significantly increase the training epochs, which is illustrated in Figure 4.3. In this subsection, we study the training time of a backbone GNN with and without being plugged into ALT. We select 4 datasets (Cora, Citeseer, Chameleon, Film) and 4 backbone GNNs (GCN, APPNP, GPRGNN, FAGCN) to show the updating time comparison per iteration in Table 4.7, which shows that ALT will increase the time of every training iteration. That is because, from Figure 4.2, we know the output of the

Table 4.7: Updating time (seconds per iteration) with and without ALT.

Backbone	ALT?	Cora	Citeseer	Chameleon	Film
GCN	No	0.0063	0.0034	0.0032	0.0029
	Yes	0.0107	0.0096	0.0094	0.0093
APPNP	No	0.0035	0.0042	0.0034	0.0031
	Yes	0.0090	0.0103	0.0089	0.0092
GPRGNN	No	0.0053	0.0045	0.0054	0.0048
	Yes	0.0121	0.0120	0.0141	0.0136
FAGCN	No	0.0044	0.0040	0.0045	0.0042
	Yes	0.0109	0.0105	0.0124	0.0115

augmenter GNN is the input of the backbone dual GNNs. Thus, according to the chain rule, the update of the augmenter GNN requires a more complex 'gradient computational graph' (and more computations) compared with the update of a vanilla backbone GNN.

4.1.5 Additional Analysis of ALT-global from the Graph Diffusion Equation (GDE) perspective

As we claimed in Lemma 4.1, our proposed ALT-global can be an adaptive filter even if the given backbone GNNs only have fixed filters. Here, we prove this from the Graph Diffusion Equation (GDE) [175] perspective. Our proof will focus on the case where the diffusion matrix is the normalized adjacency matrix $\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ whose convolution kernel is fixed. Other cases can be proved in similar ways.

Given graph signals $\mathbf{H}$, its diffusion process can be presented as $\mathbf{H}^{(t+1)} = \tilde{\mathbf{A}}\mathbf{H}^{(t)}$. Thus, we have

$$\mathbf{H}^{(t+1)} - \mathbf{H}^{(t)} = \frac{\mathbf{H}^{(t+1)} - \mathbf{H}^{(t)}}{(t+1) - t} = \tilde{\mathbf{A}}\mathbf{H}^{(t)} - \mathbf{H}^{(t)}. \tag{4.14a}$$

In the GNN case, $t > 0$ denotes the GNN depth and in the GDE context, it denotes the diffusion time. Thus, if we set the time interval as Δt, the graph diffusion dynamics can be presented as follows,

$$\frac{\mathbf{H}^{(t+1)} - \mathbf{H}^{(t)}}{\Delta t} = \tilde{\mathbf{A}}\mathbf{H}^{(t)} - \mathbf{H}^{(t)}, \quad \frac{d\mathbf{H}^{(t)}}{dt} = -\mathbf{L}\mathbf{H}^{(t)}, \tag{4.15a}$$

where $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ is the normalized Laplacian matrix. As ALT-global re-weights

all the edges into $w\tilde{\mathbf{A}}$ and $(1-w)\tilde{\mathbf{A}}$, we have

$$\frac{d\mathbf{H}_1^{(t)}}{dt} = w\tilde{\mathbf{A}}\mathbf{H}_1^{(t)} - \mathbf{H}_1^{(t)} = (-w\mathbf{L} - (1-w)\mathbf{I})\mathbf{H}_1^{(t)}, \tag{4.16a}$$

$$\frac{d\mathbf{H}_2^{(t)}}{dt} = (1-w)\tilde{\mathbf{A}}\mathbf{H}_2^{(t)} - \mathbf{H}_2^{(t)} = (-(1-w)\mathbf{L} - w\mathbf{I})\mathbf{H}_2^{(t)}, \tag{4.16b}$$

Recap that the prediction matrix of ALT-global is by combining signals from dual backbone GNNs and an offset MLP as $\mathbf{Z} = \texttt{softmax}(\mathbf{H}_1 - \mathbf{H}_2 + \eta\mathbf{H}_{\texttt{offset}})$. We keep the assumption that the dual backbone GNNs are both GA-MLPs [174], which share parameters with our offset MLP. Thus, we have $\mathbf{H}_1^{(0)} = \mathbf{H}_2^{(0)} = \mathbf{H}_{\texttt{offset}} = \mathbf{H} = \texttt{MLP}(\mathbf{X})$

As we are analyzing its diffusion dynamics, there is no interaction between any two columns of the feature matrix $\mathbf{H}_1^{(t)}$ (and $\mathbf{H}_2^{(t)}$). Hence, for brevity, we only show analysis of a single feature $\mathbf{h}_1^{(t)} = \mathbf{H}_1^{(t)}[:,m]$, $\mathbf{h}_2^{(t)} = \mathbf{H}_2^{(t)}[:,m]$, $\mathbf{h} = \mathbf{h}_{\texttt{offset}} = \mathbf{H}_{\texttt{offset}}[:,m]$, $\mathbf{z}^{(t)} = \mathbf{Z}^{(t)}[:,m]$, $\forall m \in \{1,\ldots,n\}$. The dual GNNs' GDEs can be presented as follows,

$$\frac{d\mathbf{h}_1^{(t)}}{dt} = (-w\mathbf{L} - (1-w)\mathbf{I})\mathbf{h}_1^{(t)}, \tag{4.17a}$$

$$\frac{d\mathbf{h}_2^{(t)}}{dt} = (-(1-w)\mathbf{L} - w\mathbf{I})\mathbf{h}_2^{(t)}, \tag{4.17b}$$

Proposition 4.4. The solutions of Eq. 4.17a and Eq. 4.17b can be presented as $\mathbf{h}_1^{(t)} = \sum_{i=0}^{n}\left(a_i^{(0)}e^{-(w\lambda_i+(1-w))t}\right)\mathbf{u}_i$ and $\mathbf{h}_2^{(t)} = \sum_{i=0}^{n}\left(a_i^{(0)}e^{-((1-w)\lambda_i+w)t}\right)\mathbf{u}_i$, where $\mathbf{u}_i$ and λ_i refers to the i-th eigenvector and eigenvalue of $\mathbf{L}$; initial state $a_i^{(0)}$ is determined by $\mathbf{h}_1^{(0)} = \mathbf{h}_2^{(0)} = \sum_i a_i^{(0)}\mathbf{u}_i$.

Proof. Here we prove the solution of Eq. 4.17a and for Eq. 4.17b its solution can be obtained in a similar way. For Eq. 4.17a, by decomposing the graph signal with the eigenvectors ($\{\mathbf{u}_i\}$) of the normalized Laplacian $\mathbf{L}$ we have:

$$\mathbf{h}_1^{(t)} = \sum_i a_i^{(t)}\mathbf{u}_i. \tag{4.18}$$

As only $\mathbf{h}$ and a_i are the functions of t, based on the fact that $\mathbf{L}\mathbf{u}_i = \lambda_i\mathbf{u}_i$ and $\mathbf{I}\mathbf{u}_i = \mathbf{u}_i$ we have:

$$\sum_i \left(\frac{da_i^{(t)}}{dt} + w\lambda_i a_i^{(t)} + (1-w)a_i^{(t)}\right)\mathbf{u}_i = 0. \tag{4.19}$$

As all the eigenvectors are orthogonal with each other, by multiplying both sides of the

above equation with $\mathbf{u}_i^\top$ we have

$$\left(\frac{da_i^{(t)}}{dt} + w\lambda_i a_i^{(t)} + (1-w)a_i^{(t)}\right)\mathbf{u}_i = 0. \tag{4.20}$$

$$\frac{da_i^{(t)}}{dt} + w\lambda_i a_i^{(t)} + (1-w)a_i^{(t)} = 0. \tag{4.21}$$

Hence, the graph signal $\mathbf{h}_1^{(t)}$ can be represented as

$$\mathbf{h}_1^{(t)} = \sum_{i=0}^{n}\left(a_i^{(0)}e^{-(w\lambda_i+(1-w))t}\right)\mathbf{u}_i. \tag{4.22a}$$

Similarly, the graph signal $\mathbf{h}_2^{(t)}$ can be presented as

$$\mathbf{h}_2^{(t)} = \sum_{i=0}^{n}\left(a_i^{(0)}e^{-((1-w)\lambda_i+w)t}\right)\mathbf{u}_i. \tag{4.23a}$$

QED.

Thus, the aggregated signal can be presented as

$$\mathbf{z}^{(t)} = \mathbf{h}_1^{(t)} - \mathbf{h}_2^{(t)} + \eta\mathbf{h}_{\texttt{offset}}, \tag{4.24a}$$

$$= \sum_{i=0}^{n} a_i^{(0)}\left(e^{-(w\lambda_i+(1-w))t} - e^{-((1-w)\lambda_i+w)t} + \eta\right)\mathbf{u}_i, \tag{4.24b}$$

where we use $\mathbf{h}_{\texttt{offset}} = \mathbf{h}^{(0)} = \sum_{i=1}^{n} a_i^{(0)}\mathbf{u}_i$.

According to the graph signal processing [169], $\mathbf{u}_i$ denotes the graph signal with λ_i frequency. Hence, the λ_i-frequency signal amplitude is denoted as $a_i^{(0)}\left(e^{-(w\lambda_i+(1-w))t} - e^{-((1-w)\lambda_i+w)t} + \eta\right)$ after filtered by ALT-global. We know the signal before filtering (i.e., diffusion) is

$$\mathbf{h}^{(0)} = \mathbf{h}_1^{(0)} = \mathbf{h}_2^{(0)} = \mathbf{h}_{\texttt{offset}}^{(0)} = \sum_{i=0}^{n} a_i^{(0)}\mathbf{u}_i, \tag{4.25}$$

and the amplitude of the the λ_i-frequency signal before filtering is a_i^0. Hence, the filter

response to λ_i frequency is

$$\Phi(\lambda_i) = \frac{a_i^{(0)}\left(e^{-(w\lambda_i+(1-w))t} - e^{-((1-w)\lambda_i+w)t} + \eta\right)}{a_i^{(0)}} \tag{4.26a}$$

$$= e^{-(w\lambda_i+(1-w))t} - e^{-((1-w)\lambda_i+w)t} + \eta \tag{4.26b}$$

It is clear when $w > 0$,, $\Phi(\lambda_i)$ is a monotonically decreasing function and when $w < 0$, $\Phi(\lambda_i)$ is a monotonically increasing function. With appropriate η and different w, ALT-global can be instantiated as either a low-pass filter or a high-pass filter.

4.2 DISCRETE-STATE CONTINUOUS-TIME DIFFUSION FOR GRAPH GENERATION

4.2.1 Introduction

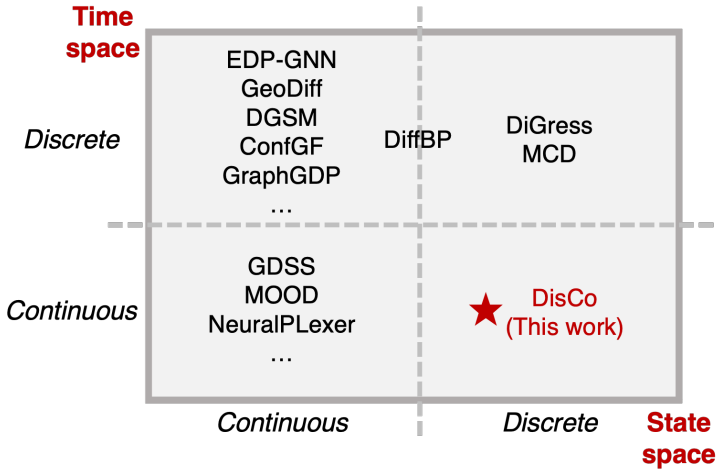


Figure 4.6: A taxonomy of graph diffusion models.

Graph generation has been studied for a long time with broad applications, based on either the one-shot (i.e., one-step) [184, 185, 186, 187, 188, 189] or auto-regressive generation paradigm [190, 191, 192, 193]. The former generates all the graph components at once and the latter does that sequentially. A recent trend of applying diffusion generative models [194, 195, 196] to graph generation tasks attracts increasing attention because of its excellent performance and solid theoretical foundation. In this paper, we follow the one-shot generation paradigm, the same as most graph diffusion generative models.

Some earlier attempts at graph diffusion models treat the graph data in a continuous

state space by viewing the graph topology and features as continuous variables [186]. Such a formulation departs from the discrete nature of graph-structured data; e.g., topological sparsity is lost, and the discretization in the generation process requires extra hyperparameters. DiGress [197] is one of the early efforts applying discrete-state diffusion models to graph generation tasks and is the current state-of-the-art graph diffusion generative model. However, DiGress is defined in the discrete time space whose generation is inflexible. This is because the number of sampling steps must match the number of forward diffusion steps, which is a fixed hyperparameter after the model finishes training. A unique advantage of the continuous-time diffusion models [189, 196] lies in their flexible sampling process, and their simulation complexity is proportional to the number of sampling steps, determined by the step size of various numerical approaches (e.g., τ-leaping [198, 199, 200]) and decoupled from the models' training. Thus, a discrete-state continuous-time diffusion model is highly desirable for graph generation tasks.

Driven by the recent advance of continuous-time Markov Chain (CTMC)-based diffusion generative model [199], we incorporate the ideas of CTMC into the corruption and denoising of graph data and propose the first discrete-state continuous-time graph diffusion generative model. It shares the same advantages as DiGress by preserving the discrete nature of graph data, while overcoming the drawback of the non-adjustable sampling process in DiGress. This Discrete-state Continuous-time graph diffusion model is named DisCo.

DisCo bears several desirable properties and advantages. First, despite its simplicity, the training objective has a rigorously proved connection to the sampling error. Second, its formulation includes a parametric graph-to-graph mapping, referred to as the backbone model, whose input-output architecture is shared between DisCo and DiGress. Therefore, the graph transformer (GT)-based backbone model [201] from DiGress can be seamlessly plugged into DisCo. Third, a concise message-passing neural network backbone model is explored with DisCo, which is simpler than the GT backbone and has decent empirical performance. Last but not least, our analyses show that the forward and reverse diffusion process in DisCo can retain the permutation-equivariant/invariant properties for its training loss and sampling distribution, both of which are critical and practical inductive biases on graph data.

Comprehensive experiments on plain and molecule graphs show that DisCo can obtain competitive or superior performance against state-of-the-art graph generative models and provide additional sampling flexibility.

4.2.2 Preliminaries

Discrete-State Continuous-Time Diffusion Models. A D-dimensional discrete state space is represented as $\mathcal{X} = \{1, \dots, C\}^D$. A continuous-time Markov Chain (CTMC) $\left\{\mathbf{x}_t = [x_t^1, \cdots x_t^D]\right\}_{t\in[0,T]}$ is characterized by its (time-dependent) rate matrix $\mathbf{R}_t \in \mathbb{R}^{|\mathcal{X}|\times|\mathcal{X}|}$. Here $\mathbf{x}_t$ is the state at the time step t. The transition probability $q_{t|s}$ between from time s to t satisfies the Kolmogorov forward equation, for $s < t$,

$$\frac{d}{dt} q_{t|s}(\mathbf{x}_t|\mathbf{x}_s) = \sum_{\xi\in\mathcal{X}} q_{t|s}(\xi|\mathbf{x}_s)\mathbf{R}_t(\xi, \mathbf{x}_t), \tag{4.27}$$

The marginal distribution can be represented as $q_t(\mathbf{x}_t) = \sum_{\mathbf{x}_0\in\mathcal{X}} q_{t|0}(\mathbf{x}_t|\mathbf{x}_0)\pi_{\texttt{data}}(\mathbf{x}_0)$

where $\pi_{\texttt{data}}(\mathbf{x}_0)$ is the data distribution. If the CTMC is defined in time interval $[0, T]$ and if the rate matrix $\mathbf{R}_t$ is well-designed, the final distribution $q_T(\mathbf{x}_T)$ can be close to a tractable reference distribution $\pi_{\texttt{ref}}(\mathbf{x}_T)$, e.g., uniform distribution. We notate the reverse stochastic process as $\tilde{\mathbf{x}}_t = \mathbf{x}_{T-t}$; a well-known fact (e.g., Section 5.9 in [202]) is that the reverse process $\{\tilde{\mathbf{x}}_t\}_{t\in[0,T]}$ is also a CTMC, characterized by the reverse rate matrix: $\tilde{\mathbf{R}}_t(\mathbf{x}, \mathbf{y}) = \frac{q(\mathbf{y})}{q(\mathbf{x})}\mathbf{R}_t(\mathbf{y}, \mathbf{x})$.

The goal of the CTMC-based diffusion models is an accurate estimation of the reverse rate matrix $\tilde{\mathbf{R}}_t$ so that new data can be generated by sampling the reference distribution $\pi_{\texttt{ref}}$ and then simulating the reverse CTMC [198, 203, 204, 205]. However, the complexity of the rate matrix is prohibitively high because there are C^D possible states. A reasonable simplification is to factorize the process over dimensions [197, 199, 200, 206]. Specifically, the forward process is factorized as $q_{t|s}(\mathbf{x}_t|\mathbf{x}_s) = \prod_{d=1}^D q_{t|s}\left(x_t^d|x_s^d\right)$, for $s < t$. Then, the forward diffusion of each dimension is independent and is governed by dimension-specific forward rate matrices $\left\{\mathbf{R}_t^d\right\}_{d=1}^D$. With such a factorization, the goal is to estimate the dimension-specific reverse rate matrices $\left\{\tilde{\mathbf{R}}_t^d\right\}_{d=1}^D$.

The dimension-specific reverse rate is represented as $\tilde{\mathbf{R}}_t^d(x^d, y^d) = \sum_{x_0^d} \mathbf{R}_t^d(y^d, x^d)\frac{q_{t|0}(y^d|x_0^d)}{q_{t|0}(x^d|x_0^d)} q_{0|t}\left(x_0^d|\mathbf{x}\right)$. Campbell et al. [199] estimate $q_{0|t}\left(x_0^d|\mathbf{x}\right)$ via a neural network p_θ such that $p_\theta(x_0^d|\mathbf{x}, t) \approx q_{0|t}\left(x_0^d|\mathbf{x}\right)$; Sun et al. [200] propose another singleton conditional distribution-based objective $\frac{p_\theta(y^d|\mathbf{x}^{\setminus d},t)}{p_\theta(x^d|\mathbf{x}^{\setminus d},t)} \approx \frac{q(y^d|\mathbf{x}^{\setminus d})}{q(x^d|\mathbf{x}^{\setminus d})}$ whose rationale is Brook's Lemma [207, 208].

Graph Generation and Notations. We study the graphs with *categorical* node and edge attributes. A graph with n nodes is represented by its edge type matrix and node type vector: $\mathcal{G} = (\mathbf{E}, \mathbf{F})$, where $\mathbf{E} = (e^{(i,j)})_{i,j\in\mathbb{N}^+_{\leq n}} \in \{1, \dots, a+1\}^{n\times n}$, $\mathbf{F} = (f^i)_{i\in\mathbb{N}^+_{\leq n}} \in \{1, \dots, b\}^n$, a and b are the numbers of node and edge types, respectively. Notably, the absence of an edge is viewed as a special edge type, so there are $(a + 1)$ edge types in total. The problem

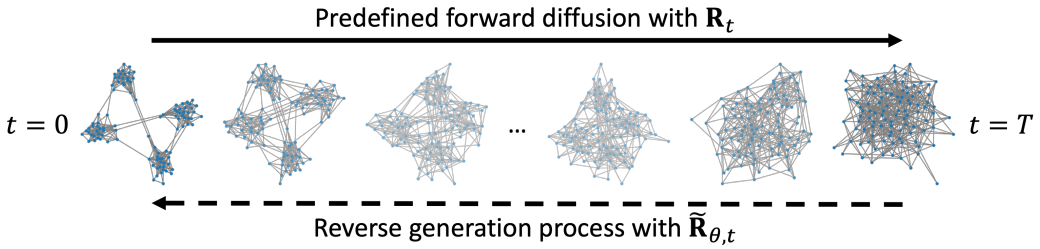


Figure 4.7: An overview of DisCo. A transition can happen at any time in $[0, T]$.

we study is graph generation where N graphs $\{\mathcal{G}^i\}_{i\in\mathbb{N}^+_{\leq N}}$ from an inaccessible graph data distribution $\mathfrak{G}$ are given and we aim to generate M graphs $\{\mathcal{G}^i\}_{i\in\mathbb{N}^+_{\leq M}}$ from $\mathfrak{G}$.

4.2.3 Method

Factorized Discrete Graph Diffusion Process. The number of possible states of an n-node graph is $(a+1)^{n^2} \times b^n$ which is intractably large. Thus, we follow existing discrete models [197, 199, 200, 206] and formulate the forward processes on every node/edge to be independent. Mathematically, the forward diffusion process for $s < t$ is factorized as

$$q_{t|s}\left(\mathcal{G}_t|\mathcal{G}_s\right) = \prod_{i,j=1}^{n} q_{t|s}\left(e_t^{(i,j)}|e_s^{(i,j)}\right) \prod_{i=1}^{n} q_{t|s}\left(f_t^i|f_s^i\right) \tag{4.28}$$

where the edge type transition probabilities $\left\{q_{t|s}\left(e_t^{(i,j)}|e_s^{(i,j)}\right)\right\}_{i,j\in\mathbb{N}^+_{\leq n}}$ and node type transition probabilities $\{q_{t|s}(f_t^i|f_s^i)\}_{i\in\mathbb{N}^+_{\leq n}}$ are characterized by their forward rate matrices $\{\mathbf{R}_t^{(i,j)}\}_{i,j\in\mathbb{N}^+_{\leq n}}$ and $\{\mathbf{R}_t^i\}_{i\in\mathbb{N}^+_{\leq n}}$, respectively. The forward processes, i.e., the forward rate matrices in our context, are predefined, which will be introduced below. Given the factorization of forward transition probability in Eq. (4.28), a question is raised: *what is the corresponding factorization of the forward rate matrix ($\mathbf{R}_t$) and the reverse rate matrix ($\tilde{\mathbf{R}}_t$)?* Remark 4.2.3 shows such a factorization.

Remark. (Factorization of rate matrices, extended from Proposition 3 of [199]) Given the

factorized forward process Eq. (4.28), the overall rate matrices are factorized as

$$\mathbf{R}_t(\bar{\mathcal{G}}, \mathcal{G}) = \sum_i A_t^i + \sum_{i,j} B_t^{(i,j)} \tag{4.29}$$

$$\tilde{\mathbf{R}}_t(\mathcal{G}, \bar{\mathcal{G}}) = \sum_i A_t^i \sum_{f_0^i} \frac{q_{t|0}\left(\bar{f}^i | f_0^i\right)}{q_{t|0}\left(f^i | f_0^i\right)} q_{0|t}\left(f_0^i | \mathcal{G}\right) + \sum_{i,j} B_t^{(i,j)} \sum_{e_0^{(i,j)}} \frac{q_{t|0}\left(\bar{e}^{(i,j)} | e_0^{(i,j)}\right)}{q_{t|0}\left(e^{(i,j)} | e_0^{(i,j)}\right)} q_{0|t}\left(e_0^{(i,j)} | \mathcal{G}\right) \tag{4.30}$$

where $A_t^i = \mathbf{R}_t^i\left(\bar{f}^i, f^i\right)\delta_{\bar{\mathcal{G}}\setminus\bar{f}^i, \mathcal{G}\setminus f^i}$, $B_t^{(i,j)} = \mathbf{R}_t^{(i,j)}\left(\bar{e}^{(i,j)}, e^{(i,j)}\right)\delta_{\bar{\mathcal{G}}\setminus\bar{e}^{(i,j)}, \mathcal{G}\setminus e^{(i,j)}}$, the operator $\delta_{\bar{\mathcal{G}}\setminus\bar{f}^i, \mathcal{G}\setminus f^i}$ (or $\delta_{\bar{\mathcal{G}}\setminus\bar{e}^{(i,j)}, \mathcal{G}\setminus e^{(i,j)}}$) checks whether two graphs $\bar{\mathcal{G}}$ and $\mathcal{G}$ are exactly the same except for node i (or the edge between nodes i and j).

Note that this factorization itself is not our contribution but a necessary part of our framework, so we mention it here for completeness. The introduction of its full derivation is postponed. Next, we detail the design of forward rate matrices.

Forward Process. A proper choice of the forward rate matrices $\left\{\mathbf{R}_t^{(i,j)}\right\}_{i,j\in\mathbb{N}_{\leq n}^+}$ and $\{\mathbf{R}_t^i\}_{i\in\mathbb{N}_{\leq n}^+}$ is important because (1) the probability distributions of node and edge types, $\{q(f_t^i)\}_{i\in\mathbb{N}_{\leq n}^+}$ and $\left\{q(e_t^{(i,j)})\right\}_{i,j\in\mathbb{N}_{\leq n}^+}$, should converge to their reference distributions within $[0, T]$ and (2) the reference distributions should be easy to sample (e.g., uniform distribution). We follow [199] to formulate $\mathbf{R}_t^{(i,j)} = \beta(t)\mathbf{R}_e^{(i,j)}$, $\forall i, j$ and $\mathbf{R}_t^i = \beta(t)\mathbf{R}_f^i$, $\forall i$, where $\beta(t)$ is a corruption schedule, $\{\mathbf{R}_e^{(i,j)}\}$ and $\{\mathbf{R}_f^i\}$ are the base rate matrices. For brevity, we set all the nodes/edges to share a common node/edge rate matrix, i.e., $\mathbf{R}_e^{(i,j)} = \mathbf{R}_e$ and $\mathbf{R}_f^i = \mathbf{R}_f$, $\forall i, j$. Then, the forward transition probability for all the nodes and edges are $q_{t|0}(f_t = v|f_0 = u) = (e^{\int_0^t \beta(s)\mathbf{R}_f ds})_{uv}$ and $q_{t|0}(e_t = v|e_0 = u) = (e^{\int_0^t \beta(s)\mathbf{R}_e ds})_{uv}$, respectively. We omit the superscript i (or (i, j)) because the transition probability is shared by all the nodes (or edges). The detailed derivation of the above analytic forward transition probability is provided in Section 4.2.8.

For categorical data, a reasonable reference distribution is a uniform distribution, i.e., $\pi_f = \frac{\mathbf{1}}{b}$ for nodes and $\pi_e = \frac{\mathbf{1}}{a+1}$ for edges. In addition, inspired by [197], we find that node and edge marginal distributions $\mathbf{m}_f$ and $\mathbf{m}_e$ are good choices as the reference distributions. Concretely, an empirical estimation of $\mathbf{m}_f$ and $\mathbf{m}_e$ is to count the number of node/edge types and normalize them. The following proposition demonstrates how to design the rate matrices to guide the forward process in converging to uniform and marginal distributions.

Proposition 4.5. The forward processes for nodes and edges converge to uniform distributions if $\mathbf{R}_f = \mathbf{1}\mathbf{1}^\top - b\mathbf{I}$ and $\mathbf{R}_e = \mathbf{1}\mathbf{1}^\top - (a+1)\mathbf{I}$; they converge to marginal distributions $\mathbf{m}_f$ and $\mathbf{m}_e$ if $\mathbf{R}_f = \mathbf{1}\mathbf{m}_f^\top - \mathbf{I}$ and $\mathbf{R}_e = \mathbf{1}\mathbf{m}_e^\top - \mathbf{I}$. $\mathbf{1}$ is an all-one vector and $\mathbf{I}$ is an identity matrix.

Regarding the selection of $\beta(t)$, we follow [195, 196, 199] and set $\beta(t) = \alpha\gamma^t \log(\gamma)$ for a smooth change of the rate matrix. α and γ are hyperparameters.

Parameterization and Optimization Objective. Next, we introduce the estimation of the reverse process from its motivation. The reverse process is essentially determined by the reverse rate matrix $\tilde{\mathbf{R}}_t$ in Eq. (4.30), whose computation needs $q_{0|t}(f_0^i|\mathcal{G})$ and $q_{0|t}(e_0^{(i,j)}|\mathcal{G})$, $\forall i, j$; their exact estimation is expensive because according to Bayes' rule, $p_t(\mathcal{G})$ is needed, whose computation needs to enumerate all the given graphs: $p_t(\mathcal{G}) = \sum_{\mathcal{G}_0} q_{t|0}(\mathcal{G}|\mathcal{G}_0)\pi_{\texttt{data}}(\mathcal{G}_0)$.

Thus, we propose parameterizing the reverse transition probabilities via a neural network θ whose specific architecture will be introduced below. The terms $\left\{q_{0|t}(f_0^i|\mathcal{G})\right\}_{i\in\mathbb{N}^+_{\leq n}}$ and $\left\{q_{0|t}(e_0^{(i,j)}|\mathcal{G})\right\}_{i,j\in\mathbb{N}^+_{\leq n}}$ in Eq. (4.30) are replaced with the parameterized $\left\{p^\theta_{0|t}(f^i|\mathcal{G})\right\}_{i\in\mathbb{N}^+_{\leq n}}$ and $\left\{p^\theta_{0|t}(e^{(i,j)}|\mathcal{G})\right\}_{i,j\in\mathbb{N}^+_{\leq n}}$. Thus, a parameterized reverse rate matrix $\tilde{\mathbf{R}}_{\theta,t}\left(\mathcal{G}, \bar{\mathcal{G}}\right)$ is represented as $\tilde{\mathbf{R}}_{\theta,t}\left(\mathcal{G}, \bar{\mathcal{G}}\right) = \sum_i \tilde{\mathbf{R}}^i_{\theta,t}\left(f^i, \bar{f}^i\right) + \sum_{i,j} \tilde{\mathbf{R}}^{(i,j)}_{\theta,t}\left(e^{(i,j)}, \bar{e}^{(i,j)}\right)$ where $\tilde{\mathbf{R}}^i_{\theta,t}\left(f^i, \bar{f}^i\right) = A^i_t \sum_{f^i_0} \frac{q_{t|0}(\bar{f}^i|f^i_0)}{q_{t|0}(f^i|f^i_0)} p^\theta_{0|t}(f^i_0|\mathcal{G})$, $\tilde{\mathbf{R}}^{(i,j)}_{\theta,t}\left(e^{(i,j)}, \bar{e}^{(i,j)}\right) = B^{(i,j)}_t \sum_{e^{(i,j)}_0} \frac{q_{t|0}(\bar{e}^{(i,j)}|e^{(i,j)}_0)}{q_{t|0}(e^{(i,j)}|e^{(i,j)}_0)} p^\theta_{0|t}\left(e^{(i,j)}_0|\mathcal{G}\right)$, and the remaining notations are the same as Eq. (4.30). Note that all the terms $\{p^\theta_{0|t}(f^i|\mathcal{G})\}_{i\in\mathbb{N}^+_{\leq n}}$ and $\{p^\theta_{0|t}(e^{(i,j)}|\mathcal{G})\}_{i,j\in\mathbb{N}^+_{\leq n}}$ can be viewed together as a graph-to-graph mapping $\theta : \mathfrak{G} \mapsto \mathfrak{G}$, whose input is the noisy graph $\mathcal{G}_t$ and its output is the predicted clean graph probabilities, concretely, the node/edge type probabilities of all the nodes and edges.

Intuitively, the discrepancy between the groundtruth $\tilde{\mathbf{R}}_t$ (from Eq. (4.30)) and the parametric $\tilde{\mathbf{R}}_{\theta,t}$ should be small. Theorem 4.1 establishes a cross-entropy (CE)-based upper bound of such a discrepancy, where the estimated probability vectors (sum is 1) are notated as $\hat{f}^i_0 = \left[p^\theta_{0|t}(f^i = 1|\mathcal{G}_t), \ldots, p^\theta_{0|t}(f^i = b|\mathcal{G}_t)\right]^\top \in [0,1]^b$ and $\hat{e}^{(i,j)}_0 = \left[p^\theta_{0|t}(e^{(i,j)} = 1|\mathcal{G}_t), \ldots, p^\theta_{0|t}(e^{(i,j)} = a+1|\mathcal{G}_t)\right]^\top \in [0,1]^{a+1}$.

Theorem 4.1 (Approximation error)**.** for $\mathcal{G} \neq \bar{\mathcal{G}}$

$$\begin{aligned}\left|\tilde{\mathbf{R}}_t(\mathcal{G}, \bar{\mathcal{G}}) - \tilde{\mathbf{R}}_{\theta,t}(\mathcal{G}, \bar{\mathcal{G}})\right|^2 \leq C_t &+ C_t^{\texttt{node}} \mathbb{E}_{\mathcal{G}_0} q_{t|0}(\mathcal{G}|\mathcal{G}_0) \sum_i \mathcal{L}_{\texttt{CE}}\left(\texttt{One-Hot}(f^i_0), \hat{f}^i_0\right) \\ &+ C_t^{\texttt{edge}} \mathbb{E}_{\mathcal{G}_0} q_{t|0}(\mathcal{G}|\mathcal{G}_0) \sum_{i,j} \mathcal{L}_{\texttt{CE}}\left(\texttt{One-Hot}(e^{(i,j)}_0), \hat{e}^{(i,j)}_0\right) \qquad (4.31)\end{aligned}$$

where C_t, $C_t^{\texttt{node}}$, and $C_t^{\texttt{edge}}$ are constants independent on θ but dependent on t, $\mathcal{G}$, and $\bar{\mathcal{G}}$; $\texttt{One-Hot}$ transforms f_0^i and $e_0^{(i,j)}$ into one-hot vectors.

The bound in Theorem 4.1 is tight, i.e., the right-hand side of Eq. (4.31) is 0, whenever $\hat{f}_0^i = q_{0|t}(f_0^i|\mathcal{G}_t), \forall i$ and $\hat{e}_0^{(i,j)} = q_{0|t}(e_0^{(i,j)}|\mathcal{G}_t), \forall i, j$. Guided by Theorem 4.1, we (1) take expectation of t by sampling t from a uniform distribution $t \sim \mathcal{U}_{(0,T)}$ and (2) simplify the right-hand side of Eq. (4.31) by using the unweighted CE loss as our training objective:

$$\min_{\theta} \; T\mathbb{E}_t\mathbb{E}_{\mathcal{G}_0}\mathbb{E}_{q_{t|0}(\mathcal{G}_t|\mathcal{G}_0)}\left[\sum_i \mathcal{L}_{\texttt{CE}}\left(\texttt{One-Hot}\left(f_0^i\right), \hat{f}_0^i\right) + \sum_{i,j} \mathcal{L}_{\texttt{CE}}\left(\texttt{One-Hot}\left(e_0^{(i,j)}\right), \hat{e}_0^{(i,j)}\right)\right] \tag{4.32}$$

A step-by-step training algorithm is in Algorithm 4.2. Note that the above CE loss has been used in some diffusion models (e.g., [199, 206]) but lacks a good motivation, especially in the continuous-time setting. We motivate it based on the rate matrix discrepancy, as a unique contribution of this paper.

Sampling Reverse Process. Given the parametric reverse rate matrix $\tilde{\mathbf{R}}_{\theta,t}(\mathcal{G}, \bar{\mathcal{G}})$, the graph generation process can be implemented by two steps: (1) sampling the reference distribution $\pi_{\texttt{ref}}$ (i.e., π_f for nodes and π_e for edges) and (2) numerically simulating the CTMC from time T to 0. The exact simulation of a CTMC has been studied for a long time, e.g., [203, 204, 205]. However, their simulation strategies only allow one transition (e.g., one edge/node type change) per step, which is highly inefficient for graphs as the number of nodes and edges is typically large; once a(n) node/edge is updated, $\tilde{\mathbf{R}}_{\theta,t}$ requires recomputation. A practical approximation is to assume $\tilde{\mathbf{R}}_{\theta,t}$ is fixed during a time interval $[t-\tau, t]$, i.e., delaying the happening of transitions in $[t-\tau, t]$ and triggering them all together at the time $t-\tau$; this strategy is also known as τ-leaping [198, 199, 200], and DisCo adopts it.

We elaborate on τ-leaping for transitions of node types; the transitions of edge types are similar. The rate matrix of the i-th node is fixed as $\tilde{\mathbf{R}}^i_{\theta,t}\left(f^i, \bar{f}^i\right) = \mathbf{R}^i_t\left(\bar{f}^i, f^i\right) \sum_{f_0^i} \frac{q_{t|0}(\bar{f}^i|f_0^i)}{q_{t|0}(f^i|f_0^i)} p^{\theta}_{0|t}(f^i|\mathcal{G}_t)$, during $[t-\tau, t]$. According to the definition of rate matrix, in $[t-\tau, t]$, the number of transitions from f^i to $\bar{f}^i$, namely $J_{f^i,\bar{f}^i}$, follows the Poisson distribution, i.e., $J_{f^i,\bar{f}^i} \sim \text{Poisson}\left(\tau \tilde{\mathbf{R}}^i_{\theta,t}\left(f^i, \bar{f}^i\right)\right)$. For categorical data (e.g., node type), multiple transitions in $[t-\tau, t]$ are invalid and meaningless. In other words, for the i-th node, if the total number of transitions $\sum_{\bar{f}^i} J_{f^i,\bar{f}^i} > 1$, f^i keeps unchanged in $[t-\tau, t]$; otherwise, if $\sum_{\bar{f}^i} J_{f^i,\bar{f}^i} = 1$ and $J_{f^i,s} = 1$, i.e., there is exact 1 transition, f^i jumps to s. A step-by-step sampling algorithm (Algorithm 4.1) is provided in Section 4.2.10.

Remark. The sampling error of τ-leaping is linear to $C_{\texttt{err}}$ [199], the approximation error of the reverse rates: $\sum_{\mathcal{G}\neq\bar{\mathcal{G}}}\left|\tilde{\mathbf{R}}_t(\mathcal{G},\bar{\mathcal{G}})-\tilde{\mathbf{R}}_{\theta,t}(\mathcal{G},\bar{\mathcal{G}})\right| \leq C_{\texttt{err}}$. Interested readers are referred to Theorem 1 from [199]. Our Theorem 4.1 shows the connection between our training loss and $C_{\texttt{err}}$, which further verifies the correctness of our training loss.

Model Instantiation. As mentioned above, the parametric backbone $p^{\theta}_{0|t}(\mathcal{G}_0|\mathcal{G}_t)$ is a graph-to-graph mapping whose input is the noisy graph $\mathcal{G}_t$ and its output is the predicted denoised graph $\mathcal{G}_0$. A wide range of neural network architectures exists. Notably, DiGress [197] uses a graph Transformer (GT) as $p^{\theta}_{0|t}$, a decent reference for our continuous-time framework. We name our model with the GT backbone as DisCo-GT. The main advantage of the GT is its long-range interaction thanks to the complete self-attention graph; however, the architecture is very complex and includes multi-head self-attention modules, leading to expensive computation.

Beyond GTs, in this paper, we posit that a regular message-passing neural network (MPNN) [209] should be a promising choice for $p^{\theta}_{0|t}(\mathcal{G}_0|\mathcal{G}_t)$. It is recognized that the MPNNs' expressiveness might not be as good as GTs' [210, 211], e.g., in terms of long-range interactions. However, in our setting, the absence of an edge is viewed as a special type of edge, and the whole graph is complete; therefore, such a limitation of MPNN is naturally mitigated, which is verified by our empirical evaluations.

Concretely, an MPNN-based graph-to-graph mapping is presented as follows, and DisCo with MPNN backbone is named DisCo-MPNN. Given a graph $\mathcal{G}=(\mathbf{E},\mathbf{F})$, where $\mathbf{E}\in\{1,\ldots,a,a+1\}^{n\times n}$, $\mathbf{F}\in\{1,\ldots,b\}^{n}$, we first transform both the matrix $\mathbf{E}$ and $\mathbf{F}$ into one-hot embeddings $\mathbf{E}_{\texttt{OH}}\in\{0,1\}^{n\times n\times(a+1)}$ and $\mathbf{F}_{\texttt{OH}}\in\{0,1\}^{n\times b}$. Then, some auxiliary features (e.g., the # of specific motifs) are extracted: $\mathbf{F}_{\texttt{aux}},\mathbf{y}_{\texttt{aux}}=\texttt{Aux}(\mathbf{E}_{\texttt{OH}})$ to overcome the expressiveness limitation of MPNNs [212]. Here $\mathbf{F}_{\texttt{aux}}$ and $\mathbf{y}_{\texttt{aux}}$ are the node and global auxiliary features, respectively. Note that a similar auxiliary feature engineering is also applied in DiGress [197]. More details about the `Aux` can be found in Section 4.2.11. Then, three multi-layer perceptrons (MLPs) are used to map node features $\mathbf{F}_{\texttt{OH}}\oplus\mathbf{F}_{\texttt{aux}}$, edge features $\mathbf{E}_{\texttt{OH}}$, and global features $\mathbf{y}_{\texttt{aux}}$ into a common hidden space as $\mathbf{F}_{\texttt{hidden}}=\texttt{MLP}(\mathbf{F}_{\texttt{OH}}\oplus\mathbf{F}_{\texttt{aux}})$, $\mathbf{E}_{\texttt{hidden}}=\texttt{MLP}(\mathbf{E}_{\texttt{OH}})$, $\mathbf{y}_{\texttt{hidden}}=\texttt{MLP}(\mathbf{y}_{\texttt{aux}})$, where $\oplus$ is a concatenation operator. The following formulas present the update of node embeddings (e.g., $\mathbf{r}^{i}=\mathbf{F}(i,:)$), edge embedding (e.g., $\mathbf{r}^{(i,j)}=\mathbf{E}(i,j,:)$), and global embedding $\mathbf{y}$ in an MPNN layer, where we omit the subscript

`hidden` if it does not cause ambiguity:

$$\mathbf{r}^i \leftarrow \texttt{FiLM}\left(\texttt{FiLM}\left(\mathbf{r}^i, \texttt{MLP}\left(\sum_{j=1}^{n} \mathbf{r}^{(j,i)}/n\right)\right), \mathbf{y}\right), \tag{4.33}$$

$$\mathbf{r}^{(i,j)} \leftarrow \texttt{FiLM}\left(\texttt{FiLM}\left(\mathbf{r}^{(i,j)}, \mathbf{r}^i \odot \mathbf{r}^j\right), \mathbf{y}\right), \tag{4.34}$$

$$\mathbf{y} \leftarrow \mathbf{y} + \texttt{PNA}\left(\{\mathbf{r}^i\}_{i=1}^n\right) + \texttt{PNA}\left(\{\mathbf{r}^{(i,j)}\}_{i,j=1}^n\right). \tag{4.35}$$

The edge embeddings are aggregated by mean pooling (i.e., $\sum_{j=1}^{n} \mathbf{r}^{(j,i)}/n$); the node pair embeddings are passed to edges by Hadamard product (i.e., $\mathbf{r}^i \odot \mathbf{r}^j$); edge/node embeddings are merged to the global embedding $\mathbf{y}$ via the PNA module [213]; Some FiLM modules [214] are used for the interaction between node/edge/global embeddings. More details about the PNA and FiLM are in Section 4.2.11. In this paper, we name Eqs. (4.34) and (4.35) on all nodes/edges together as an MPNN layer, $\mathbf{F}, \mathbf{E}, \mathbf{y} \leftarrow \texttt{MPNN}(\mathbf{F}, \mathbf{E}, \mathbf{y})$. Stacking multiple MPNN layers results in a larger model capacity. Finally, two readout MLPs are used to project the node/edge embeddings into input dimensions, $\texttt{MLP}(\mathbf{F}) \in \mathbb{R}^{n \times b}$ and $\texttt{MLP}(\mathbf{E}) \in \mathbb{R}^{n \times n \times (a+1)}$, which are output after wrapped with `softmax`.

Both the proposed MPNN and the GT from DiGress [197] use the PNA and FiLM to merge embeddings, but MPNN does not have multi-head self-attention layers, so the computation overhead is lower.

Permutation Equivariance and Invariance. Reordering the nodes keeps the property of a given graph, which is known as permutation invariance. In addition, for a given function, if its input is permuted and its output is permuted accordingly, such a behavior is known as permutation equivariance. In this subsection, we analyze permutation-equivariance/invariance of the (1) diffusion framework (Lemmas 4.3, 4.4, and 4.5), (2) sampling density (Theorem 4.2), and (3) training loss (Theorem 4.3).

Lemma 4.3 (Permutation-equivariant layer)**.** The proposed MPNN layer (Eqs. (4.34) and (4.35)) is permutation-equivariant.

The auxiliary features from the `Aux` are also permutation-equivariant (see Section 4.2.11). Thus, the whole MPNN-based backbone $p_{0|t}^{\theta}$ is permutation-equivariant. Note that the GT-based backbone from DiGress [197] is also permutation-equivariant, whose proof is omitted as it is not our contribution. Next, we show the permutation invariance of the rate matrices.

Lemma 4.4 (Permutation-invariant rate matrices)**.** The forward rate matrix of DisCo is permutation-invariant if it is factorized as Eq. (4.29). The parametric reverse rate ma-

trix of DisCo ($\tilde{\mathbf{R}}_{\theta,t}$) is permutation-invariant whenever the graph-to-graph backbone $p^{\theta}_{0|t}$ is permutation-equivariant.

Lemma 4.5 (Permutation-invariant transition probability)**.** For CTMC satisfying the Kolmogorov forward equation (Eq. (4.27)), if the rate matrix is permutation-invariant (i.e., $\mathbf{R}_t(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{R}_t(\mathcal{P}(\mathbf{x}_i), \mathcal{P}(\mathbf{x}_j))$, the transition probability is permutation-invariant (i.e., $q_{t|s}(\mathbf{x}_t|\mathbf{x}_s) = q_{t|s}(\mathcal{P}(\mathbf{x}_t)|\mathcal{P}(\mathbf{x}_s))$, where $\mathcal{P}$ is a permutation.

Based on Lemmas 4.4 and 4.5, DisCo's parametric reverse transition probability is permutation-invariant. The next theorem shows the permutation-invariance of the sampling probability.

Theorem 4.2 (Permutation-invariant sampling probability)**.** If both the reference distribution $\pi_{\texttt{ref}}$ and the reverse transition probability are permutation-invariant, the parametric sampling distribution $p^{\theta}_0(\mathcal{G}_0)$ is permutation-invariant.

In addition, the next theorem shows the permutation invariance of the training loss.

Theorem 4.3 (Permutation-invariant training loss)**.** The proposed training loss Eq. (4.32) is invariant to any permutation of the input graph $\mathcal{G}_0$ if $p^{\theta}_{0|t}$ is permutation-equivariant.

4.2.4 Experiments: Plain Graph Generation

Datasets. Datasets SBM, Planar [187], and Community [190] are used. We follow the settings of SPECTRE [187] and DiGress [197] to split the SBM, Planar [187], and Community [190] datasets into 64/16/20% for training/validation/test sets.

Table 4.8: Dataset statistics.

Name	# Graphs	Split	a	b	Avg. |E|	Max |E|	Avg. |F|	Max |F|
SBM	200	128/32/40	1	1	1000.8	2258	104.0	187
Planar	200	128/32/40	1	1	355.7	362	64.0	64
Community	100	64/16/20	1	1	74.0	122	15.7	20
QM9	130831	97734/20042/13055	4	4	18.9	28	8.8	9
MOSES	1733214	1419512/156176/157526	4	8	46.3	62	21.6	27
GuacaMol	1398213	1118633/69926/209654	4	12	60.4	176	27.8	88

Metrics. The Maximum Mean Discrepancy (MMD) [190] measures the discrepancy between two sets of distributions. The relative squared MMD [197] is defined as follows

$$score = \frac{\mathrm{MMD}^2(\{\mathcal{G}\}_{\texttt{gen}}||\{\mathcal{G}\}_{\texttt{test}})}{\mathrm{MMD}^2(\{\mathcal{G}\}_{\texttt{train}}||\{\mathcal{G}\}_{\texttt{test}})}, \tag{4.36}$$

where $(\{\mathcal{G}\}_{\texttt{gen}}$, $(\{\mathcal{G}\}_{\texttt{train}}$, and $(\{\mathcal{G}\}_{\texttt{test}}$ are the sets of generated graphs, training graphs, and test graphs, respectively. We report the above relative squared MMD for degree distributions (Deg.), clustering coefficient distributions (Clus.), and average orbit counts (Orb.) statistics (the number of occurrences of all substructures with 4 nodes). In addition, uniqueness, novelty, and validity were chosen. Uniqueness reports the fraction of the generated nonisomorphic graphs; Novelty reports the fraction of the generated graphs not isomorphic with any graph from the training set; Validity checks the fraction of the generated graphs following some specific rules. For the SBM dataset, we follow the validity check from [187], whose core idea is to check whether real SBM graphs are statistically indistinguishable from the generated graphs; for the Planar dataset, we check whether the generated graphs are connected and are indeed planar graphs. Since the Community dataset lacks the Validity metric, we only report the Uniqueness, Novelty, and Validity results for the SBM and Planar datasets.

We report mean±std in 5 runs.

Baseline Methods. GraphRNN [190], GRAN [192], GG-GAN [215], MolGAN [216], SPECTRE [187], EDP-GNN [186], GraphGDP [217], DiscDDPM [218], EDGE [219], ConGress [197], DiGress [197] are chosen.

Results. Table 4.9 shows the effectiveness evaluation on SBD and Planar from which we observe:

- DisCo-GT can obtain competitive performance against the SOTA, DiGress, which is reasonable because both models share the graph Transformer backbone. Note that DiGress's performance in terms of Validity is not the statistics reported in the paper but from their latest model checkpoint [3]. In fact, we found it very hard for DiGress and DisCo-GT to learn to generate valid SBM/Planar graphs. These two datasets have only 200 graphs, but sometimes only after $> 10,000$ epochs training, the Validity percentage can be $> 50\%$. Additionally, DisCo-GT provides extra flexibility during sampling by adjusting the τ. This is important: our models can still trade-off between the sampling efficiency and quality even after the model is trained and frozen.

- In general, DisCo-MPNN has competitive performance against DisCo-GT in terms of Deg., Clus., and Orb. However, its performance is worse compared to DisCo-GT in terms of Validity, which might be related to the different model expressiveness.

[3] `https://github.com/cvignac/DiGress/blob/main/README.md`

Table 4.9: Performance (mean±std) on SBM and Planar datasets.

Dataset	Model	Deg.↓	Clus.↓	Orb.↓	Unique ↑	Novel ↑	Valid ↑
SBM	GraphRNN [190]	6.9	1.7	3.1	**100.0**	**100.0**	5.0
	GRAN [192]	14.1	1.7	2.1	**100.0**	**100.0**	25.0
	GG-GAN [215]	4.4	2.1	2.3	**100.0**	**100.0**	0.0
	MolGAN [216]	29.4	3.5	2.8	95.0	**100.0**	10.0
	SPECTRE [187]	1.9	1.6	**1.6**	**100.0**	**100.0**	52.5
	ConGress [197]	34.1	3.1	4.5	0.0	0.0	0.0
	DiGress [197]	1.6	1.5	1.7	**100.0**	**100.0**	**67.5**
	DisCo-MPNN	1.8±0.2	**0.8±0.1**	2.7±0.4	**100.0±0.0**	**100.0±0.0**	41.9±2.2
	DisCo-GT	**0.8±0.2**	**0.8±0.4**	2.0±0.5	**100.0±0.0**	**100.0±0.0**	66.2±1.4
Planar	GraphRNN [190]	24.5	9.0	2508.0	**100.0**	**100.0**	0.0
	GRAN [192]	3.5	1.4	1.8	85.0	2.5	**97.5**
	GG-GAN [215]	315.0	8.3	2062.6	**100.0**	**100.0**	0.0
	MolGAN [216]	4.5	10.2	2346.0	25.0	**100.0**	0.0
	SPECTRE [187]	2.5	2.5	2.4	**100.0**	**100.0**	25.0
	ConGress [197]	23.8	8.8	2590.0	0.0	0.0	0.0
	DiGress [197]	1.4	**1.2**	**1.7**	**100.0**	**100.0**	85.0
	DisCo-MPNN	1.4±0.3	1.4±0.4	6.4±1.6	**100.0±0.0**	**100.0±0.0**	33.8±2.7
	DisCo-GT	**1.2**±0.5	1.3±0.5	**1.7±0.7**	**100.0±0.0**	**100.0±0.0**	83.6±2.1

Studying the expressiveness of the graph-to-graph model would be an interesting future direction, e.g., generating valid Planar graphs.

4.2.5 Experiments: Molecule Graph Generation

Datasets. The datasets QM9 [225], MOSES [226], and GuacaMol [227] are chosen. We follow the split of QM9 from DiGress [197] and follow the split of MOSES [226] and GuacaMol [227] according to their benchmark settings. Their statistics are presented in Table 4.8.

Metrics. For QM9, Uniqueness, Novelty, and Validity are chosen as metrics. The first two are the same as introduced above. The Validity is computed by building a molecule with RdKit [4] and checking if we can obtain a valid SMILES string from it.

For MOSES, the chosen metrics include Uniqueness, Novelty, Validity, Filters, Frchet ChemNet Distance (FCD), Similarity to a nearest neighbor (SNN), and Scaffold similarity (Scaf), which is consistent with DiGress [197]. The official evaluation code [5] is used to report the performance.

[4] `https://www.rdkit.org/`

[5] `https://github.com/molecularsets/moses`

Table 4.10: Performance (mean±std%) on QM9 dataset. V., U., and N. mean Valid, Unique, and Novel.

Model	Valid ↑	V.U. ↑	V.U.N. ↑
CharacterVAE [220]	10.3	7.0	6.3
GrammarVAE[221]	60.2	5.6	4.5
GraphVAE [222]	55.7	42.0	26.1
GT-VAE [223]	74.6	16.8	15.8
Set2GraphVAE [188]	59.9	56.2	-
GG-GAN [215]	51.2	24.4	24.4
MolGAN [216]	98.1	10.2	9.6
SPECTRE [187]	87.3	31.2	29.1
GraphNVP [184]	83.1	82.4	-
GDSS [189]	95.7	94.3	-
EDGE [219]	99.1	**99.1**	-
ConGress [197]	98.9	95.7	38.3
DiGress [197]	99.0	95.2	31.8
GRAPHARM [224]	90.3	86.3	-
DisCo-MPNN	98.9±0.7	98.7±0.5	**68.7±0.2**
DisCo-GT	**99.3±0.6**	98.9±0.6	56.2±0.4

For GuacaMol, the chosen metrics include Uniqueness, Novelty, Validity, KL Divergence, and Frehet ChemNet Distance (FCD), which is consistent with DiGress [197]. The official evaluation code[6] is used to report the performance.

We report mean±std in 5 runs except MOSES and GuacaMol, whose computations are too expensive to repeat multiple times.

Baseline Methods. CharacterVAE [220], GrammarVAE [221], GraphVAE [222], GT-VAE [223], Set2GraphVAE [188], GG-GAN [215], MolGAN [216], SPECTRE [187], GraphNVP [184], GDSS [189], EDGE [219], ConGress [197], DiGress [197], GRAPHARM [224], VAE [228], JT-VAE [191], GraphINVENT [229], LSTM [230], NAGVAE [231], and MCTS [232] are chosen.

Results. Table 4.10 shows the performance on QM9 dataset. Our observation is consistent with the performance comparison on plain datasets: (1) DisCo-GT obtains slightly better or at least competitive performance against DiGress due to the shared graph-to-graph backbone, but our framework offers extra flexibility in the sampling process; (2) DisCo-MPNN obtains decent performance in terms of Validity, Uniqueness, and Novelty comparing with

[6] `https://github.com/BenevolentAI/guacamol`

Table 4.11: Performance on MOSES. VAE, JT-VAE, and GraphINVENT have hard-coded rules to ensure high validity.

Model	Valid ↑	Unique ↑	Novel ↑	Filters ↑	FCD ↓	SNN ↑	Scaf ↑
VAE [228]	97.7	98.8	69.5	99.7	0.57	0.58	5.9
JT-VAE [191]	100.0	100.0	99.9	97.8	1.00	0.53	10.0
GraphINVENT [229]	96.4	99.8	N/A	95.0	1.22	0.54	12.7
ConGress [197]	83.4	99.9	96.4	94.8	1.48	0.50	16.4
DiGress [197]	85.7	100.0	95.0	97.1	1.19	0.52	14.8
DisCo-MPNN	83.9	100.0	98.8	87.3	1.63	0.48	13.5
DisCo-GT	88.3	100.0	97.7	95.6	1.44	0.50	15.1

Table 4.12: Performance on GuacaMol. LSTM, NAGVAE, and MCTS are tailored for molecule datasets; ConGress, DiGress, and DisCo are general graph generation models.

Model	Valid ↑	Unique ↑	Novel ↑	KL div ↑	FCD ↑
LSTM [230]	95.9	100.0	91.2	99.1	91.3
NAGVAE [231]	92.9	95.5	100.0	38.4	0.9
MCTS [232]	100.0	100.0	95.4	82.2	1.5
ConGress [197]	0.1	100.0	100.0	36.1	0.0
DiGress [197]	85.2	100.0	99.9	92.9	68.0
DisCo-MPNN	68.7	100.0	96.4	77.0	36.4
DisCo-GT	86.6	100.0	99.9	92.6	59.7

DisCo-GT.

Tables 4.11 and 4.12 show the performance on MOSES and GuacaMol, which further verifies that (1) performance of DisCo-GT is on par with the SOTA general graph generative models, DiGress, and (2) DisCo-MPNN has decent performance, but worse than DisCo-GT and DiGress.

Table 4.13: Efficiency comparison in terms of number of parameters, forward and backpropagation time (seconds/iteration).

Backbone	**GT**	**MPNN**
# Parameters	14×10^6	7×10^6
Forward	0.065	0.022
Backprop.	0.034	0.018

Table 4.14: Ablation study (mean±std%) with GT backbone. V., U., and N. mean Valid, Unique, and Novel.

Reference Dist.	**Steps**	**Valid ↑**	**V.U. ↑**	**V.U.N. ↑**
Marginal	500	99.3±0.6	98.9±0.6	56.2±0.4
	100	98.7±0.5	98.5±0.4	58.8±0.4
	30	97.9±1.2	97.6±1.1	59.2±0.8
	10	95.3±1.9	94.8±1.6	62.1±0.9
	5	93.0±1.7	92.4±1.3	64.9±1.1
	1	76.1±2.3	73.9±1.6	62.9±1.8
Uniform	500	94.1±0.9	92.9±0.5	56.6±0.4
	100	91.5±1.0	90.3±0.9	54.4±1.2
	30	88.7±1.6	86.9±1.0	58.6±2.1
	10	84.5±2.3	80.4±1.7	59.8±1.8
	5	77.0±2.5	69.9±1.5	56.1±3.5
	1	44.9±3.1	35.1±3.4	29.6±2.5

4.2.6 Additional Experiments

Efficiency Study. A major computation bottleneck is the graph-to-graph backbone $p^{\theta}_{0|t}$, which is GT or MPNN. We compare the number of parameters, the forward and backpropagation time of GT and MPNN in Table 4.13. For a fair comparison, we set all the hidden dimensions of GT and MPNN as 256 and the number of layers as 5. We use the Community [190] dataset and set the batch size as 64. Table 4.13 shows that GT has a larger capacity and more parameters at the expense of more expensive training.

Ablation Study. An ablation study on DisCo-GT for reference distributions (marginal vs. uniform), and sampling steps (1 to 500) is presented in Table 4.14. The number of sampling steps is $\texttt{round}(\frac{1}{\tau})$ if $T = 1$. QM9 dataset is chosen. A similar ablation study on DisCo-MPNN is in Table 4.16. We observe that, first, generally, the fewer sampling steps, the lower the generation quality. In some cases (e.g., the marginal distribution), with the sampling steps decreasing significantly (e.g., from 500 to 30), the performance degradation

Table 4.15: Generation performance (mean±std) on the Community dataset.

Model	Deg.↓	Clus.↓	Orb.↓
GraphRNN [190]	4.0	1.7	4.0
GRAN [192]	3.0	1.6	1.0
EDP-GNN [186]	2.5	2.0	3.0
GraphGDP [217]	2.0	1.1	-
DiscDDPM [218]	1.2	**0.9**	1.5
EDGE [219]	1.0	1.0	2.0
GG-GAN [215]	4.0	3.1	8.0
MolGAN [216]	3.0	1.9	1.0
SPECTRE [187]	0.5	2.7	2.0
DiGress [197]	1.0	**0.9**	1.0
DisCo-MPNN	1.4±0.5	**0.9±0.2**	**0.9±0.3**
DisCo-GT	**0.9±0.2**	**0.9±0.3**	1.1±0.4

remains very slight, indicating our method's high robustness to changes in sampling steps. Second, the marginal reference distribution is better than the uniform distribution, consistent with the observation from DiGress [197].

Additional Results on Community Additional Community plain graph dataset results are in Table 4.15. Our observation is consistent with the main content: both variants of DisCo are on par with, or even better than, the SOTA general graph diffusion generative model, DiGress.

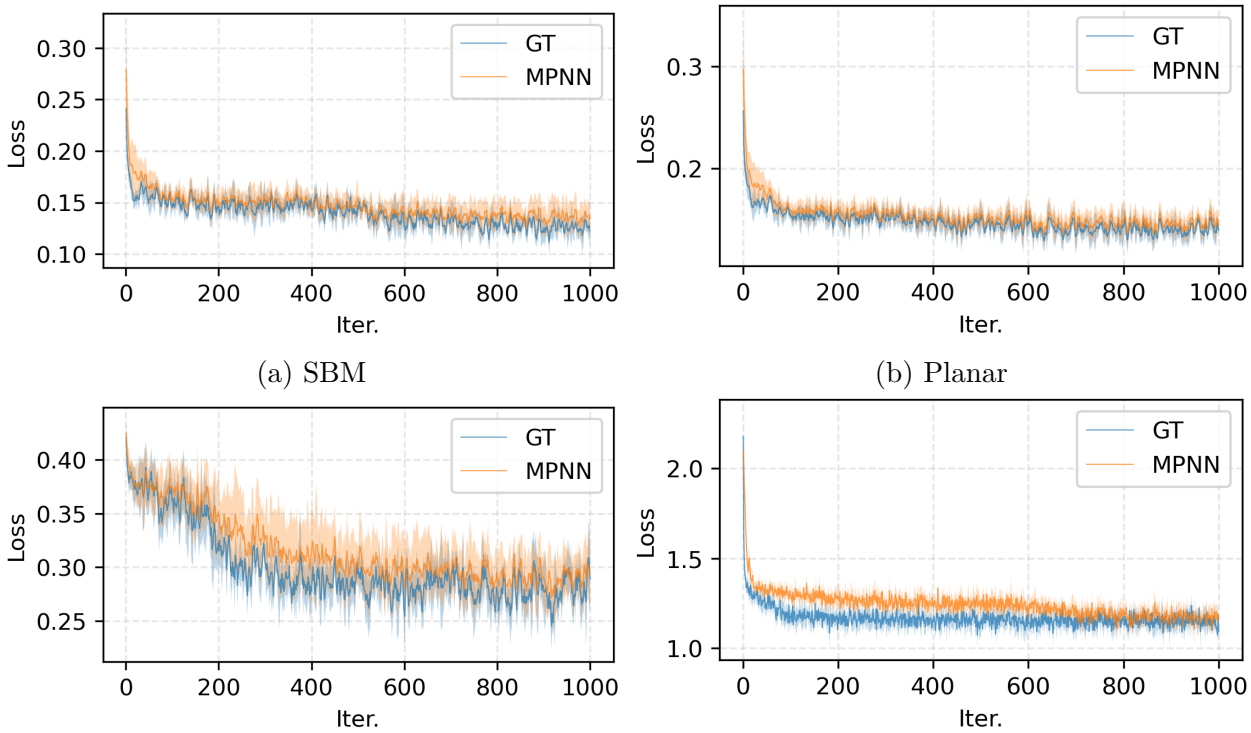
Additional Ablation Study Table 4.16 shows the ablation study of DisCo-MPNN on QM9 dataset. Our observations are consistent with the main content: (1) generally, the fewer sampling steps, the lower the generation quality but method's performance is robust in terms of the decreasing of sampling steps; (2) the marginal reference distribution is better than the uniform distribution, consistent with the observation from DiGress [197].

Convergence Study. Figure 4.8 shows the training loss of DisCo-GT and DisCo-MPNN on four datasets, whose X-axis is the number of iterations (i.e., the number of epochs $\times$ the number of training samples / batch size). We found that overall the training losses converge smoothly on 4 datasets.

Visualization. The generated graphs on the SBM and Planar datasets are presented in Figure 4.9. We clarify that the generated planar graphs are *selected to be valid* because, as

Table 4.16: Ablation study (mean±std%) with MPNN backbone. V., U., and N. mean Valid, Unique, and Novel.

Reference Dist.	Steps	Valid ↑	V.U. ↑	V.U.N. ↑
Marginal	500	98.9±0.7	98.7±0.5	68.7±0.2
	100	98.4±1.1	98.0±1.0	69.1±0.6
	30	97.7±1.2	97.5±0.8	70.4±1.1
	10	92.3±1.9	91.9±2.2	66.4±1.7
	5	88.8±3.3	87.1±2.8	67.3±2.9
	1	64.4±2.7	63.2±1.9	55.8±1.4
Uniform	500	93.5±1.7	93.2±1.1	64.9±1.0
	100	93.1±2.1	92.6±1.7	66.2±1.9
	30	87.1±1.8	86.8±1.1	64.0±1.0
	10	83.7±3.2	81.9±2.1	61.3±2.0
	5	81.5±2.9	75.4±3.4	64.6±2.3
	1	71.3±2.3	42.2±4.0	36.9±3.2



(a) SBM

(b) Planar

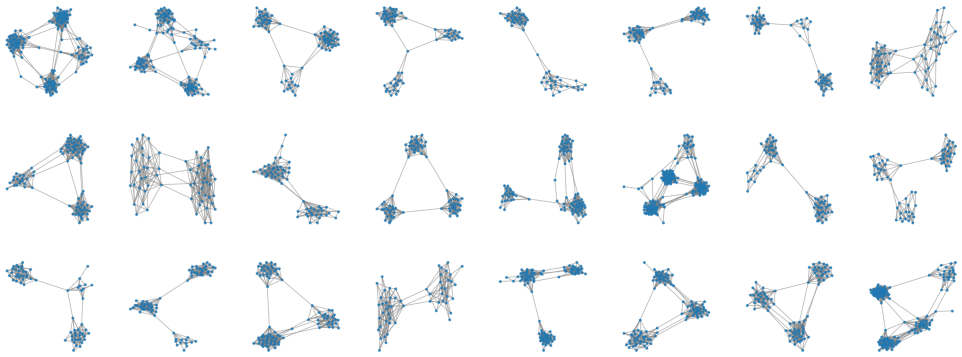
(c) Community

(d) QM9

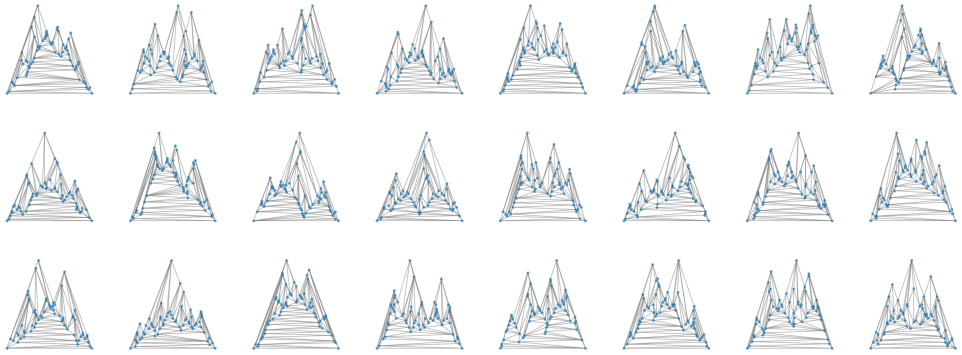
Figure 4.8: Training loss of DisCo on different datasets and backbone models.

Table 4.9 shows, not all the generated graphs are valid planar graphs, but the planar layout

can only visualize valid planar graphs in our setting [7]. The generated SBM graphs are not selected; even if a part of them cannot pass the strict SBM statistic test, most, if not all, of them still form $2-5$ densely connected clusters.



(a) SBM



(b) Planar

Figure 4.9: Generated graphs.

4.2.7 Details of the Factorization of Rate Matrices

In this section, we detail the derivation of the factorization of the rate matrix, which is extended from the following Proposition 3 of [199].

[7] `https://networkx.org/documentation/stable/reference/generated/networkx.drawing.layout.planar_layout.html`

Proposition 4.6 (Factorization of the rate matrix, Proposition 3 from [199])**.** If the forward process factorizes as $q_{t|s}(\mathbf{x}_t|\mathbf{x}_s) = \prod_{d=1}^{D} q_{t|s}(x_t^d|x_s^d), t > s$, then the forward and reverse rates are of the form

$$\mathbf{R}_t(\bar{\mathbf{x}}, \mathbf{x}) = \sum_{d=1}^{D} \mathbf{R}_t^d(\bar{x}^d, x^d)\delta_{\bar{\mathbf{x}}\backslash\bar{x}^d, \mathbf{x}\backslash x^d} \tag{4.37}$$

$$\tilde{\mathbf{R}}_t(\mathbf{x}, \bar{\mathbf{x}}) = \sum_{d=1}^{D} \mathbf{R}_t^d(\bar{x}^d, x^d)\delta_{\bar{\mathbf{x}}\backslash\bar{x}^d, \mathbf{x}\backslash x^d} \sum_{x_0^d} q_{0|t}(x_0^d|\mathbf{x}) \frac{q_{t|0}(\bar{x}^d|x_0^d)}{q_{t|0}(x^d|x_0^d)} \tag{4.38}$$

where $\delta_{\bar{\mathbf{x}}\backslash\bar{x}^d, \mathbf{x}\backslash x^d} = 1$ when all dimensions except for d are equal.

As all the nodes and edges are categorical, applying the above proposition of all the nodes and edges leads to our Remark 4.2.3.

4.2.8 Details of Forward Transition Probability

In this section, we present the derivation of the forward transition probability for nodes; the forward process for edges can be derived similarly. Note that this derivation has been mentioned in [199] for generic discrete cases; we graft it to the graph settings and include it here for completeness. The core derivation of the forward transition probability is to prove the following proposition.

Proposition 4.7 (Analytical forward process for commutable rate matrices, Proposition 10 from [199])**.** if $\mathbf{R}_t$ and $\mathbf{R}_{t'}$ commute $\forall t, t'$, $q_{t|0}(x_t = j|x_0 = i) = \left(e^{\int_0^t \mathbf{R}_s ds}\right)_{ij}$

Proof. If $q_{t|0} = \exp\left(\int_0^t \mathbf{R}_s ds\right)$ is the forward transition probability matrix, it should satisfy the Kolmogorov forward equation $\frac{d}{dt} q_{t|0} = q_{t|0}\mathbf{R}_s$. The transition probability matrix

$$q_{t|0} = \sum_{k=0}^{\infty} \frac{1}{k!} \left(\int_0^t \mathbf{R}_s ds\right)^k, \tag{4.39}$$

and, based on the fact that $\mathbf{R}_t$ and $\mathbf{R}_t'$ commute $\forall t, t'$, its derivative is

$$\frac{d}{dt} q_{t|0} = \sum_{k=1}^{\infty} \frac{1}{(k-1)!} \left(\int_0^t \mathbf{R}_s ds\right)^{(k-1)} = q_{t|0}\mathbf{R}_t. \tag{4.40}$$

Thus, $q_{t|0} = \exp\left(\int_0^t \mathbf{R}_s ds\right)$ is the solution of Kolmogorov forward equation. QED.

For the node i, if its forward rate matrix is set as $\mathbf{R}_t^i = \beta(t)\mathbf{R}_f$, we have $\mathbf{R}_t^i$ and $\mathbf{R}_{t'}^i$ commute, $\forall t, t'$. Thus, the transition probability for node i is $q_{t|0}\left(f_t^i = v | f_0^i = u\right) = \left(e^{\int_0^t \beta(s)\mathbf{R}_f ds}\right)_{uv}$. Based on similar derivation, we have the transition probability for the edge (i, j) as $q_{t|0}\left(e_t^{(i,j)} = v | e_0^{(i,j)} = u\right) = \left(e^{\int_0^t \beta(s)\mathbf{R}_e ds}\right)_{uv}$.

4.2.9 Proofs

Proof of Proposition 4.5. Proposition 4.5 claims the forward process converges to uniform distributions if $\mathbf{R}_f = \mathbf{1}\mathbf{1}^\top - b\mathbf{I}$ and $\mathbf{R}_e = \mathbf{1}\mathbf{1}^\top - (a+1)\mathbf{I}$ and it converges to marginal distributions $\mathbf{m}_f$ and $\mathbf{m}_e$ if $\mathbf{R}_f = \mathbf{1}\mathbf{m}_f^\top - \mathbf{I}$ and $\mathbf{R}_e = \mathbf{1}\mathbf{m}_e^\top - \mathbf{I}$.

Proof. If we formulate the rate matrices for nodes and edges as $\mathbf{R}_t^{(i,j)} = \beta(t)\mathbf{R}_e$, $\forall i, j$ and $\mathbf{R}_t^i = \beta(t)\mathbf{R}_f$, $\forall i$, every rate matrix is commutable for any time steps t and t'. In the following content, we show the proof for the node rate matrix $\mathbf{R}_t^i = \beta(t)\mathbf{R}_f$; the converged distribution of edge can be proved similarly. Based on Proposition 4.7, the transition probability matrix between time steps t and $t + \Delta t$ is

$$q_{t+\Delta t|t} = \mathbf{I} + \int_t^{t+\Delta t} \beta(s)\mathbf{R}_f ds + O((\Delta t)^2) \tag{4.41}$$

$$\overset{(*)}{=} \mathbf{I} + \Delta t \beta(\xi)\mathbf{R}_f + O((\Delta t)^2), \tag{4.42}$$

where (*) is based on the Mean Value Theorem. If the high-order term $O((\Delta t)^2)$ is omitted and we short $\beta_{\Delta t} = \Delta t \beta(\xi)$, for $\mathbf{R}_f = \mathbf{1}\mathbf{1}^\top - b\mathbf{I}$, we have

$$q_{t+\Delta t|t} \approx \beta_{\Delta t}\mathbf{1}\mathbf{1}^\top + (1 - \beta_{\Delta t} b)\mathbf{I}, \tag{4.43}$$

which is the transition matrix of the uniform diffusion in the discrete-time diffusion models [194, 206, 233]. Thus, with $T \to \infty$ and $q_{t+\Delta t|t}$ to the power of infinite, the converged distribution is a uniform distribution. Similarly, for $\mathbf{R}_f = \mathbf{1}\mathbf{m}_f^\top - \mathbf{I}$ the transition matrix is

$$q_{t+\Delta t|t} \approx \beta_{\Delta t}\mathbf{1}\mathbf{m}_f^\top + (1 - \beta_{\Delta t})\mathbf{I} \tag{4.44}$$

which is a generalized transition matrix of the 'absorbing state' diffusion [206]. The difference lies at for the 'absorbing state' diffusion [206], $\mathbf{m}_f$ is set as a one-hot vector for the absorbing state, and here we set it as the marginal distribution. Thus, with $T \to \infty$ and $q_{t+\Delta t|t}$ to the power of infinite, the converged distribution is a marginal distribution $\mathbf{m}_f$. QED.

Proof of Theorem 4.1. Theorem 4.1 says for $\mathcal{G} \neq \bar{\mathcal{G}}$,

$$\left|\tilde{\mathbf{R}}_t(\mathcal{G}, \bar{\mathcal{G}}) - \tilde{\mathbf{R}}_{\theta,t}(\mathcal{G}, \bar{\mathcal{G}})\right|^2 \leq C_t + C_t^{\texttt{node}} \mathbb{E}_{\mathcal{G}_0} q_{t|0}(\mathcal{G}|\mathcal{G}_0) \sum_i \mathcal{L}_{\texttt{CE}}\left(\texttt{One-Hot}\left(f_0^i\right), \hat{f}_0^i\right) + C_t^{\texttt{edge}} \mathbb{E}_{\mathcal{G}_0} q_{t|0}(\mathcal{G}|\mathcal{G}_0) \sum_{i,j} \mathcal{L}_{\texttt{CE}}\left(\texttt{One-Hot}\left(e_0^{(i,j)}\right), \hat{e}_0^{(i,j)}\right) \tag{4.45}$$

where the node and edge estimated probability vector (sum is 1) is notated as $\hat{f}_0^i = [p_{0|t}^\theta(f^i = 1|\mathcal{G}_t), \ldots, p_{0|t}^\theta(f^i = b|\mathcal{G}_t)]^\top \in [0,1]^b$ and $\hat{e}_0^{(i,j)} = [p_{0|t}^\theta(e^{(i,j)} = 1|\mathcal{G}_t), \ldots, p_{0|t}^\theta(e^{(i,j)} = a{+}1|\mathcal{G}_t)]^\top \in [0,1]^{a+1}$.

Proof.

$$\left|\tilde{\mathbf{R}}_t(\mathcal{G}, \bar{\mathcal{G}}) - \tilde{\mathbf{R}}_{\theta,t}(\mathcal{G}, \bar{\mathcal{G}})\right| \tag{4.46}$$

$$= \Big| \sum_i A_t^i \sum_{f_0^i} \frac{q_{t|0}(\bar{f}^i|f_0^i)}{q_{t|0}(f^i|f_0^i)} (q_{0|t}(f_0^i|\mathcal{G}) - p_{0|t}^\theta(f_0^i|\mathcal{G})) + \sum_{i,j} B_t^{(i,j)} \sum_{e_0^{(i,j)}} \frac{q_{t|0}(\bar{e}^{(i,j)}|e_0^{(i,j)})}{q_{t|0}(e^{(i,j)}|e_0^{(i,j)})} (q_{0|t}(e_0^{(i,j)}|\mathcal{G}) - p_{0|t}^\theta(e_0^{(i,j)}|\mathcal{G}))\Big| \tag{4.47}$$

$$\leq \Big| \sum_i A_t^i \sum_{f_0^i} \frac{q_{t|0}(\bar{f}^i|f_0^i)}{q_{t|0}(f^i|f_0^i)} (q_{0|t}(f_0^i|\mathcal{G}) - p_{0|t}^\theta(f_0^i|\mathcal{G}))\Big| + \Big| \sum_{i,j} B_t^{(i,j)} \sum_{e_0^{(i,j)}} \frac{q_{t|0}(\bar{e}^{(i,j)}|e_0^{(i,j)})}{q_{t|0}(e^{(i,j)}|e_0^{(i,j)})} (q_{0|t}(e_0^{(i,j)}|\mathcal{G}) - p_{0|t}^\theta(e_0^{(i,j)}|\mathcal{G}))\Big| \tag{4.48}$$

We check the first term of Eq. (4.48):

$$\Big|\sum_i A_t^i \sum_{f_0^i} \frac{q_{t|0}(\bar{f}^i|f_0^i)}{q_{t|0}(f^i|f_0^i)}(q_{0|t}(f_0^i|\mathcal{G}) - p^\theta_{0|t}(f_0^i|\mathcal{G}))\Big| \tag{4.49}$$

$$\leq \sum_i A_t^i \sup_{f_0^i}\Big\{\frac{q_{t|0}(\bar{f}^i|f_0^i)}{q_{t|0}(f^i|f_0^i)}\Big\} \sum_{f_0^i}\Big|q_{0|t}(f_0^i|\mathcal{G}) - p^\theta_{0|t}(f_0^i|\mathcal{G})\Big| \tag{4.50}$$

$$= \sum_i C_i \sum_{f_0^i}\Big|q_{0|t}(f_0^i|\mathcal{G}) - p^\theta_{0|t}(f_0^i|\mathcal{G})\Big| \tag{4.51}$$

$$\overset{(*)}{\leq} \sum_i C_i \sqrt{2\sum_{f_0^i}\Big(C_{f_0^i} - q_{0|t}(f_0^i|\mathcal{G})\log p^\theta_{0|t}(f_0^i|\mathcal{G})\Big)} \tag{4.52}$$

$$\overset{(**)}{\leq} C_1 \sqrt{\sum_i\sum_{f_0^i}\Big(C_{f_0^i} - q_{0|t}(f_0^i|\mathcal{G})\log p^\theta_{0|t}(f_0^i|\mathcal{G})\Big)} \tag{4.53}$$

$$= C_1 \sqrt{C_2 - \sum_i\sum_{f_0^i} q_{0|t}(f_0^i|\mathcal{G})\log p^\theta_{0|t}(f_0^i|\mathcal{G})} \tag{4.54}$$

where $C_i = A_t^i \sup_{f_0^i}\Big\{\frac{q_{t|0}(\bar{f}^i|f_0^i)}{q_{t|0}(f^i|f_0^i)}\Big\}$, $C_{f_0^i} = q_{0|t}(f_0^i|\mathcal{G})\log q_{0|t}(f_0^i|\mathcal{G})$, (*) is based on the Pinsker's inequality, (**) is based on CauchySchwarz inequality: $\sum_{i=1}^n \sqrt{x_i} \leq \sqrt{n\sum_{i=1}^n x_i}$, $C_1 = \sqrt{2n}\sup_i\{C_i\}$, $C_2 = \sum_i \sum_{f_0^i} C_{f_0^i}$. Next, the term $-\sum_i\sum_{f_0^i} q_{0|t}(f_0^i|\mathcal{G})\log p^\theta_{0|t}(f_0^i|\mathcal{G})$ is equivalent to:

$$-\sum_i\sum_{f_0^i} q_{0|t}(f_0^i|\mathcal{G})\log p^\theta_{0|t}(f_0^i|\mathcal{G}) \tag{4.55}$$

$$= -\frac{1}{p_t(\mathcal{G})}\sum_i\sum_{f_0^i} p_{0,t}(f_0^i,\mathcal{G})\log p^\theta_{0|t}(f_0^i|\mathcal{G}) \tag{4.56}$$

$$= -\frac{1}{p_t(\mathcal{G})}\sum_i\sum_{f_0^i}\sum_{\mathcal{G}_0(f_0^i)} p_{0,t}(\mathcal{G}_0,\mathcal{G})\log p^\theta_{0|t}(f_0^i|\mathcal{G}) \tag{4.57}$$

$$= -\frac{1}{p_t(\mathcal{G})}\sum_i\sum_{f_0^i}\sum_{\mathcal{G}_0(f_0^i)} \pi_{\texttt{data}}(\mathcal{G}_0)q_{t|0}(\mathcal{G}|\mathcal{G}_0)\log p^\theta_{0|t}(f_0^i|\mathcal{G}) \tag{4.58}$$

$$= \frac{1}{p_t(\mathcal{G})}\sum_i\sum_{f_0^i}\sum_{\mathcal{G}_0(f_0^i)} \pi_{\texttt{data}}(\mathcal{G}_0)q_{t|0}(\mathcal{G}|\mathcal{G}_0)\mathcal{L}_{\texttt{CE}}(\texttt{One-Hot}(f_0^i),\hat{f}_0^i) \tag{4.59}$$

$$= \frac{1}{p_t(\mathcal{G})}\sum_{\mathcal{G}_0} \pi_{\texttt{data}}(\mathcal{G}_0)q_{t|0}(\mathcal{G}|\mathcal{G}_0)\sum_i \mathcal{L}_{\texttt{CE}}(\texttt{One-Hot}(f_0^i),\hat{f}_0^i) \tag{4.60}$$

$$= \frac{1}{p_t(\mathcal{G})}\mathbb{E}_{\mathcal{G}_0} q_{t|0}(\mathcal{G}|\mathcal{G}_0)\sum_i \mathcal{L}_{\texttt{CE}}(\texttt{One-Hot}(f_0^i),\hat{f}_0^i) \tag{4.61}$$

where $\sum_{\mathcal{G}_0(f_0^i)}$ marginalizing all the graphs at time 0 whose i-th node is f_0^i; $p_{0,t}(f_0^i, \mathcal{G})$ is the joint probability of a graph whose i-th node is f_0^i at time 0 and it is $\mathcal{G}$ at time t; $p_{0,t}(\mathcal{G}_0, \mathcal{G})$ is the joint probability of a graph which is $\mathcal{G}_0$ at time 0 and it is $\mathcal{G}$ at time t. Plugging Eq. (4.61) into Eq. (4.54):

$$\begin{aligned}&\left|\sum_i A_t^i \sum_{f_0^i} \frac{q_{t|0}(\bar{f}^i|f_0^i)}{q_{t|0}(f^i|f_0^i)}(q_{0|t}(f_0^i|\mathcal{G}) - p_{0|t}^\theta(f_0^i|\mathcal{G}))\right| \\ \leq & C_1\sqrt{C_2 + C_5\mathbb{E}_{\mathcal{G}_0} q_{t|0}(\mathcal{G}|\mathcal{G}_0)\sum_i \mathcal{L}_{\texttt{CE}}(\texttt{One-Hot}(f_0^i), \hat{f}_0^i)}\end{aligned} \tag{4.62}$$

where $C_5 = \frac{1}{p_t(\mathcal{G})}$. A similar analysis can be conducted about the second term of Eq. (4.48) and we directly present it here:

$$\begin{aligned}&\left|\sum_{i,j} B_t^{(i,j)} \sum_{e_0^{(i,j)}} \frac{q_{t|0}(\bar{e}^{(i,j)}|e_0^{(i,j)})}{q_{t|0}(e^{(i,j)}|e_0^{(i,j)})}(q_{0|t}(e_0^{(i,j)}|\mathcal{G}) - p_{0|t}^\theta(e_0^{(i,j)}|\mathcal{G}))\right| \\ \leq & C_3\sqrt{C_4 + C_5\mathbb{E}_{\mathcal{G}_0} q_{t|0}(\mathcal{G}|\mathcal{G}_0)\sum_{i,j} \mathcal{L}_{\texttt{CE}}(\texttt{One-Hot}(e_0^{(i,j)}), \hat{e}_0^{(i,j)})}\end{aligned} \tag{4.63}$$

where $C_3 = \sqrt{2}n\sup_{i,j}\{C_{i,j}\}$, $C_4 = \sum_{i,j}\sum_{e_0^{(i,j)}} C_{e_0^{(i,j)}}$, $C_{i,j} = B_t^{(i,j)}\sup_{e_0^{(i,j)}}\left\{\frac{q_{t|0}(\bar{e}^{(i,j)}|e_0^{(i,j)})}{q_{t|0}(e^{(i,j)}|e_0^{(i,j)})}\right\}$, $C_{e_0^{(i,j)}} = q_{0|t}(e_0^{(i,j)}|\mathcal{G})\log q_{0|t}(e_0^{(i,j)}|\mathcal{G})$.

Plugging Eqs. (4.62) and (4.63) into Eq. (4.48), being aware that C_1, C_2, C_3, C_4, C_5 are all t-related:

$$\begin{aligned}\left|\tilde{\mathbf{R}}_t(\mathcal{G}, \bar{\mathcal{G}}) - \tilde{\mathbf{R}}_{\theta,t}(\mathcal{G}, \bar{\mathcal{G}})\right| \leq & C_1\sqrt{C_2 + C_5\mathbb{E}_{\mathcal{G}_0} q_{t|0}(\mathcal{G}|\mathcal{G}_0)\sum_i \mathcal{L}_{\texttt{CE}}(\texttt{One-Hot}(f_0^i), \hat{f}_0^i)} \\ & + C_3\sqrt{C_4 + C_5\mathbb{E}_{\mathcal{G}_0} q_{t|0}(\mathcal{G}|\mathcal{G}_0)\sum_{i,j} \mathcal{L}_{\texttt{CE}}(\texttt{One-Hot}(e_0^{(i,j)}), \hat{e}_0^{(i,j)})} \qquad (4.64) \\ \overset{(*)}{\leq} & \Big(C_t + C_t^{\texttt{node}}\mathbb{E}_{\mathcal{G}_0} q_{t|0}(\mathcal{G}|\mathcal{G}_0)\sum_i \mathcal{L}_{\texttt{CE}}(\texttt{One-Hot}(f_0^i), \hat{f}_0^i) \\ & + C_t^{\texttt{edge}}\mathbb{E}_{\mathcal{G}_0} q_{t|0}(\mathcal{G}|\mathcal{G}_0)\sum_{i,j} \mathcal{L}_{\texttt{CE}}(\texttt{One-Hot}(e_0^{(i,j)}), \hat{e}_0^{(i,j)})\Big)^{1/2} \qquad (4.65)\end{aligned}$$

where (*) is based on CauchySchwarz inequality, $C_t = 2C_1^2C_2 + 2C_3^2C_4$, $C_t^{\texttt{node}} = 2C_1^2C_5$, $C_t^{\texttt{edge}} = 2C_3^2C_5$. QED.

Proof of Lemma 4.3. We clarify that the term "permutation" in this paper refers to the reordering of the node indices, i.e., the first dimension of $\mathbf{F}$ and the first two dimensions of $\mathbf{E}$.

Proof. The input of an MPNN layer is $\mathbf{F} = \{\mathbf{r}_i\}_{i=1}^n \in \mathbb{R}^{n\times d}, \mathbf{E} = \{\mathbf{r}_{i,j}\}_{i,j=1}^n \in \mathbb{R}^{n\times n\times d}, \mathbf{y} \in \mathbb{R}^d$, where d is the hidden dimension. The updating formulas of an MPNN layer can be presented as

$$\mathbf{r}^i \leftarrow \texttt{FiLM}\left(\texttt{FiLM}\left(\mathbf{r}^i, \texttt{MLP}\left(\frac{\sum_{j=1}^n \mathbf{r}^{(j,i)}}{n}\right)\right), \mathbf{y}\right), \tag{4.66}$$

$$\mathbf{r}^{(i,j)} \leftarrow \texttt{FiLM}\left(\texttt{FiLM}(\mathbf{r}^{(i,j)}, \mathbf{r}^i \odot \mathbf{r}^j), \mathbf{y}\right), \tag{4.67}$$

$$\mathbf{y} \leftarrow \mathbf{y} + \texttt{PNA}\left(\{\mathbf{r}^i\}_{i=1}^n\right) + \texttt{PNA}\left(\{\mathbf{r}^{(i,j)}\}_{i,j=1}^n\right), \tag{4.68}$$

The permutation $\mathcal{P}$ of the input of an MPNN layer can be presented as $\mathcal{P}\left(\mathbf{F} = \{\mathbf{r}_i\}_{i=1}^n, \mathbf{E} = \{\mathbf{r}_{i,j}\}_{i,j=1}^n, \mathbf{y}\right) = \left(\{\mathbf{r}_{\sigma(i)}\}_{i=1}^n, \{\mathbf{r}_{\sigma(i),\sigma(j)}\}_{i,j=1}^n, \mathbf{y}\right)$ where $\sigma : \{1,\dots,n\} \mapsto \{1,\dots,n\}$ is a bijection.

For `PNA` (Eq. (4.98)), it includes operations `max`, `min`, `mean`, and `std` which are all permutation-invariant and thus, the `PNA` module is permutation-invariant. Then,

$$\mathbf{y} + \texttt{PNA}\big(\{\mathbf{r}^i\}_{i=1}^n\big) + \texttt{PNA}\big(\{\mathbf{r}^{(i,j)}\}_{i,j=1}^n\big) = \mathbf{y} + \texttt{PNA}\big(\{\mathbf{r}^{\sigma(i)}\}_{i=1}^n\big) + \texttt{PNA}\big(\{\mathbf{r}^{(\sigma(i),\sigma(j))}\}_{i,j=1}^n\big) \tag{4.69}$$

Because $\sum_{j=1}^n \mathbf{r}^{(j,i)} = \sum_{j=1}^n \mathbf{r}^{(\sigma(j),\sigma(i))}$, $\mathbf{r}^i \odot \mathbf{r}^j = \mathbf{r}^{\sigma(i)} \odot \mathbf{r}^{\sigma(j)}$, and the FiLM module (Eq. (4.99)) is not related to the node ordering,

$$\mathbf{r}^{(\sigma(i),\sigma(j))} \leftarrow \texttt{FiLM}\left(\texttt{FiLM}\left(\mathbf{r}^{(\sigma(i),\sigma(j))}, \mathbf{r}^{\sigma(i)} \odot \mathbf{r}^{\sigma(j)}\right), \mathbf{y}\right) = \texttt{FiLM}\left(\texttt{FiLM}\left(\mathbf{r}^{(i,j)}, \mathbf{r}^i \odot \mathbf{r}^j\right), \mathbf{y}\right) \tag{4.70}$$

$$\mathbf{r}^{\sigma(i)} \leftarrow \texttt{FiLM}\left(\texttt{FiLM}\left(\mathbf{r}^{\sigma(i)}, \texttt{MLP}\left(\frac{\sum_{j=1}^n \mathbf{r}^{(\sigma(j),\sigma(i))}}{n}\right)\right), \mathbf{y}\right) \tag{4.71}$$

$$= \texttt{FiLM}\left(\texttt{FiLM}\left(\mathbf{r}^i, \texttt{MLP}\left(\frac{\sum_{j=1}^n \mathbf{r}^{(j,i)}}{n}\right)\right), \mathbf{y}\right) \tag{4.72}$$

Thus, we proved that

$$\texttt{MPNN}\left(\mathcal{P}\left(\mathbf{F}, \mathbf{E}, \mathbf{y}\right)\right) = \mathcal{P}\left(\texttt{MPNN}(\mathbf{F}, \mathbf{E}, \mathbf{y})\right) \tag{4.73}$$

QED.

Proof of Lemma 4.4.

Proof. The forward rate matrix (Eq. (4.29)) is the sum of component-specific forward rate matrices ($\{\mathbf{R}_t^{(i,j)}\}_{i,j\in\mathbb{N}^+_{\leq n}}$ and $\{\mathbf{R}_t^i\}_{i\in\mathbb{N}^+_{\leq n}}$). It is permutation-invariant because the summation is permutation-invariant.

The parametric reverse rate matrix is

$$\tilde{\mathbf{R}}_{\theta,t}(\mathcal{G},\bar{\mathcal{G}}) = \sum_i \tilde{\mathbf{R}}^i_{\theta,t}(f^i,\bar{f}^i) + \sum_{i,j} \tilde{\mathbf{R}}^{(i,j)}_{\theta,t}(e^{(i,j)},\bar{e}^{(i,j)}) \tag{4.74}$$

where $\tilde{\mathbf{R}}^i_{\theta,t}(f^i,\bar{f}^i) = A^i_t \sum_{f^i_0} \frac{q_{t|0}(\bar{f}^i|f^i_0)}{q_{t|0}(f^i|f^i_0)} p^\theta_{0|t}(f^i_0|\mathcal{G}_t)$,
$\tilde{\mathbf{R}}^{(i,j)}_{\theta,t}(e^{(i,j)},\bar{e}^{(i,j)}) = B^{(i,j)}_t \sum_{e^{(i,j)}_0} \frac{q_{t|0}(\bar{e}^{(i,j)}|e^{(i,j)}_0)}{q_{t|0}(e^{(i,j)}|e^{(i,j)}_0)} p^\theta_{0|t}(e^{(i,j)}_0|\mathcal{G}_t)$. If we present the permutation $\mathcal{P}$ on every node as a bijection $\sigma:\{1,\dots,n\}\mapsto\{1,\dots,n\}$, the term

$$\begin{aligned}
\tilde{\mathbf{R}}^i_{\theta,t}(f^i,\bar{f}^i) &= A^i_t \sum_{f^i_0} \frac{q_{t|0}(\bar{f}^i|f^i_0)}{q_{t|0}(f^i|f^i_0)} p^\theta_{0|t}(f^i_0|\mathcal{G}_t) && (4.75)\\
&= \mathbf{R}^i_t(\bar{f}^i,f^i)\delta_{\bar{\mathcal{G}}\setminus\bar{f}^i,\mathcal{G}\setminus f^i} \sum_{f^i_0} \frac{q_{t|0}(\bar{f}^i|f^i_0)}{q_{t|0}(f^i|f^i_0)} p^\theta_{0|t}(f^i_0|\mathcal{G}_t) && (4.76)\\
&\overset{(*)}{=} \mathbf{R}^{\sigma(i)}_t(\bar{f}^{\sigma(i)},f^{\sigma(i)})\delta_{\mathcal{P}(\bar{\mathcal{G}})\setminus\bar{f}^{\sigma(i)},\mathcal{P}(\mathcal{G})\setminus f^{\sigma(i)}} \sum_{f^{\sigma(i)}_0} \frac{q_{t|0}(\bar{f}^{\sigma(i)}|f^{\sigma(i)}_0)}{q_{t|0}(f^{\sigma(i)}|f^{\sigma(i)}_0)} p^\theta_{0|t}(f^i_0|\mathcal{G}_t) && (4.77)\\
&\overset{(**)}{=} \mathbf{R}^{\sigma(i)}_t(\bar{f}^{\sigma(i)},f^{\sigma(i)})\delta_{\mathcal{P}(\bar{\mathcal{G}})\setminus\bar{f}^{\sigma(i)},\mathcal{P}(\mathcal{G})\setminus f^{\sigma(i)}} \sum_{f^{\sigma(i)}_0} \frac{q_{t|0}(\bar{f}^{\sigma(i)}|f^{\sigma(i)}_0)}{q_{t|0}(f^{\sigma(i)}|f^{\sigma(i)}_0)} p^\theta_{0|t}(f^{\sigma(i)}_0|\mathcal{P}(\mathcal{G}_t)) && (4.78)\\
&= \tilde{\mathbf{R}}^{\sigma(i)}_{\theta,t}(f^{\sigma(i)},\bar{f}^{\sigma(i)}) && (4.79)
\end{aligned}$$

where (*) is based on the permutation invariant of the forward process and its rate matrix; (**) is based on the permutation equivariance of the graph-to-graph backbone $p^\theta_{0|t}$. QED.

Proof of Lemma 4.5. Recall the Kolmogorov forward equation, for $s<t$,

$$\frac{d}{dt} q_{t|s}(\mathbf{x}_t|\mathbf{x}_s) = \sum_{\xi\in\mathcal{X}} q_{t|s}(\xi|\mathbf{x}_s)\mathbf{R}_t(\xi,\mathbf{x}_t). \tag{4.80}$$

Proof. We aim to show that $q_{t|s}(\mathcal{P}(\mathbf{x}_t)|\mathcal{P}(\mathbf{x}_s))$ is a solution of Eq. (4.80). Because the

permutation $\mathcal{P}$ is a bijection, we have

$$\frac{d}{dt}q_{t|s}(\mathcal{P}(\mathbf{x}_t)|\mathcal{P}(\mathbf{x}_s)) \tag{4.81}$$

$$=\sum_{\xi\in\mathcal{X}} q_{t|s}(\mathcal{P}(\xi)|\mathcal{P}(\mathbf{x}_s))\mathbf{R}_t(\mathcal{P}(\xi),\mathcal{P}(\mathbf{x}_t)) \tag{4.82}$$

$$\overset{(*)}{=}\sum_{\xi\in\mathcal{X}} q_{t|s}(\mathcal{P}(\xi)|\mathcal{P}(\mathbf{x}_s))\mathbf{R}_t(\xi,\mathbf{x}_t) \tag{4.83}$$

where (*) is because $\mathbf{R}_t$ is permutation-invariant. As Eq. (4.83) and Eq. (4.80) share the same rate matrix, and the rate matrix completely determines the CTMC (and its Kolmogorov forward equation) [202], thus, their solutions are the same: $q_{t|s}(\mathbf{x}_t|\mathbf{x}_s) = q_{t|s}(\mathcal{P}(\mathbf{x}_t)|\mathcal{P}(\mathbf{x}_s))$, i.e., the transition probability is permutation-invariant. QED.

Proof of Theorem 4.2.

Proof. We start from a simple case where the parametric rate matrix is fixed all the time,

$$p_0^\theta(\mathcal{G}_0) = \sum_{\mathcal{G}_T} q_{0|T}^\theta(\mathcal{G}_0|\mathcal{G}_T)\pi_{\texttt{ref}}(\mathcal{G}_T), \tag{4.84}$$

where the transition probability is by solving the Kolmogorov forward equation

$$\frac{d}{dt}q_{t|s}^\theta(\mathcal{G}_t|\mathcal{G}_s) = \sum_{\xi} q_{t|s}^\theta(\xi|\mathcal{G}_s)\tilde{\mathbf{R}}_\theta(\xi,\mathcal{G}_t). \tag{4.85}$$

Thus, the sampling probability of permuted graph $\mathcal{P}(\mathcal{G}_0)$

$$p_0^\theta(\mathcal{P}(\mathcal{G}_0)) = \sum_{\mathcal{G}_T} q_{0|T}^\theta(\mathcal{P}(\mathcal{G}_0)|\mathcal{P}(\mathcal{G}_T))\pi_{\texttt{ref}}(\mathcal{P}(\mathcal{G}_T)) \tag{4.86}$$

$$\overset{(*)}{=} \sum_{\mathcal{G}_T} q_{0|T}^\theta(\mathcal{G}_0|\mathcal{G}_T)\pi_{\texttt{ref}}(\mathcal{P}(\mathcal{G}_T)) \tag{4.87}$$

$$\overset{(**)}{=} \sum_{\mathcal{G}_T} q_{0|T}^\theta(\mathcal{G}_0|\mathcal{G}_T)\pi_{\texttt{ref}}(\mathcal{G}_T) \tag{4.88}$$

$$= p_0^\theta(\mathcal{G}_0) \tag{4.89}$$

where (*) is based on Lemma 4.4 and Lemma 4.5, the transition probability of DisCo is permutation-invariant and (**) is from the assumption that the reference distribution $\pi_{\texttt{ref}}(\mathcal{G}_T)$ is permutation-invariant. Thus, we proved that for the simple case, $\tilde{\mathbf{R}}_{\theta,t}$ fixed $\forall t$, the sampling probability is permutation-invariant.

For the practical sampling, as we mentioned in Section 4.2.3, the τ-leaping algorithm assumes that the time interval $[0, T]$ is divided into various length-τ intervals $[0, \tau), [\tau, 2\tau), \ldots, [T-\tau, T]$ (here both close sets or open sets work) and assume the reverse rate matrix is fixed as $\tilde{\mathbf{R}}_{\theta,t}$ within every length-τ interval, such as $(t-\tau, t]$. Thus, the sampling probability can be computed as

$$p_0^\theta(\mathcal{G}_0) = \sum_{\mathcal{G}_T, \mathcal{G}_{T-\tau}, \ldots, \mathcal{G}_\tau} q_{0|\tau}(\mathcal{G}_0|\mathcal{G}_\tau) \ldots q_{T-\tau|T}(\mathcal{G}_{T-\tau}|\mathcal{G}_T)\pi_{\texttt{ref}}(\mathcal{G}_T). \tag{4.90}$$

The conclusion from the simple case can be generalized to this τ-leaping-based case because all the transition probabilities $q_{t-\tau|t}(\mathcal{G}_{t-\tau}|\mathcal{G}_t)$ and the reference distribution are permutation-invariant. QED.

Note that Xu et al. [234] have a similar analysis in their Proposition 1 on a DDPM-based model.

Proof of Theorem 4.3. Recall our training objective is

$$\min_\theta T\mathbb{E}_{t\sim\mathcal{U}_{(0,T)}}\mathbb{E}_{\mathcal{G}_0}\mathbb{E}_{q_{t|0}(\mathcal{G}_t|\mathcal{G}_0)}\left[\sum_i \mathcal{L}_{\texttt{CE}}(\texttt{One-Hot}(f_0^i), \hat{f}_0^i) + \sum_{i,j}\mathcal{L}_{\texttt{CE}}(\texttt{One-Hot}(e_0^{(i,j)}), \hat{e}_0^{(i,j)})\right] \tag{4.91}$$

where $\hat{f}_0^i = [p_{0|t}^\theta(f^i = 1|\mathcal{G}_t), \ldots, p_{0|t}^\theta(f^i = b|\mathcal{G}_t)]^\top \in [0,1]^b$ and
$\hat{e}_0^{(i,j)} = [p_{0|t}^\theta(e^{(i,j)} = 1|\mathcal{G}_t), \ldots, p_{0|t}^\theta(e^{(i,j)} = a+1|\mathcal{G}_t)]^\top \in [0,1]^{a+1}$

Proof. We follow the notation and present the permutation $\mathcal{P}$ on every node as a bijection $\sigma : \{1, \ldots, n\} \mapsto \{1, \ldots, n\}$. We first analyze the cross-entropy loss on the nodes for a single training graph $\mathcal{G}_0$ and taking expectation $\mathbb{E}_{\mathcal{G}_0}$ keeps the permutation invariance:

$$\begin{aligned}
\mathcal{L}_{\texttt{node}}(\mathcal{G}_0) &= T\mathbb{E}_{t\sim\mathcal{U}_{(0,T)}}\mathbb{E}_{q_{t|0}(\mathcal{G}_t|\mathcal{G}_0)}\sum_i \mathcal{L}_{\texttt{CE}}(\texttt{One-Hot}(f_0^i), \hat{f}_0^i) && (4.92)\\
&= T\mathbb{E}_{t\sim\mathcal{U}_{(0,T)}}\sum_{\mathcal{G}_t} q_{t|0}(\mathcal{G}_t|\mathcal{G}_0)\sum_i \mathcal{L}_{\texttt{CE}}(\texttt{One-Hot}(f_0^i), \hat{f}_0^i) && (4.93)\\
&\overset{(*)}{=} T\mathbb{E}_{t\sim\mathcal{U}_{(0,T)}}\sum_{\mathcal{G}_t} q_{t|0}(\mathcal{P}(\mathcal{G}_t)|\mathcal{P}(\mathcal{G}_0))\sum_i \mathcal{L}_{\texttt{CE}}(\texttt{One-Hot}(f_0^i), \hat{f}_0^i) && (4.94)\\
&\overset{(**)}{=} T\mathbb{E}_{t\sim\mathcal{U}_{(0,T)}}\sum_{\mathcal{G}_t} q_{t|0}(\mathcal{P}(\mathcal{G}_t)|\mathcal{P}(\mathcal{G}_0))\sum_i \mathcal{L}_{\texttt{CE}}(\texttt{One-Hot}(f_0^{\sigma(i)}), \hat{f}_0^{\sigma(i)}) && (4.95)\\
&= \mathcal{L}_{\texttt{node}}(\mathcal{P}(\mathcal{G}_0)) && (4.96)
\end{aligned}$$

where (*) is from the permutation invariance of the forward process and (**) is from the permutation equivariance of the graph-to-graph backbone and the permutation invariance of the cross-entropy loss. A similar result can be analyzed on the cross-entropy loss on the edges

$$\mathcal{L}_{\texttt{edge}}(\mathcal{G}_0) = T\mathbb{E}_{t\sim\mathcal{U}_{(0,T)}}\mathbb{E}_{q_{t|0}(\mathcal{G}_t|\mathcal{G}_0)}\sum_{i,j}\mathcal{L}_{\texttt{CE}}(\texttt{One-Hot}(e_0^{(i,j)}), \hat{e}_0^{(i,j)}) = \mathcal{L}_{\texttt{edge}}(\mathcal{P}(\mathcal{G}_0)) \tag{4.97}$$

and we omit the proof here for brevity. QED.

4.2.10 Sampling Algorithm

A Step-by-step procedure about the τ-leaping graph generation is presented in Algorithm 4.1.

4.2.11 Auxiliary Features, PNA and FiLM Modules

For learning a better graph-to-graph mapping $p^{\theta}_{0|t}(\mathcal{G}_0|\mathcal{G}_t)$, artificially augmenting the node-level features and graph-level features is proved effective to enhance the expressiveness of graph learning models [197, 235]. For this setting, we keep consistent with the state-of-the-art model, DiGress [197], and extract the following three sets of auxiliary features. Note that the following features are extracted on the noisy graph $\mathcal{G}_t$.

We binarize the edge tensor $\mathbf{E}$ into an adjacency matrix $\mathbf{A} \in \{0,1\}^{n\times n}$ whose 1 entries denote that any type of edge connects the corresponding node pair.

Motif Features. The number of length-3/4/5 cycles every node is included in is counted as the topological node-level features; also, the total number of length-3/4/5/6 cycles is the topological graph-level feature.

Spectral Features. The graph Laplacian is decomposed. The number of connected components and the first 5 non-zero eigenvalues are selected as the spectral graph-level features. An estimated indicator of whether a node is included in the largest connected component, and the first 2 eigenvectors of the non-zero eigenvalues are selected as the spectral node-level features.

Molecule Features. On molecule datasets, the valency of each atom is selected as the node-level feature, and the total weight of the whole molecule is selected as the graph-level

Algorithm 4.1: τ-Leaping Graph Generation

$t \leftarrow T$;

$\mathcal{G}_t = (\{e^{(i,j)}\}_{i,j \in \mathbb{N}^+_{\leq n}}, \{f^i\}_{i \in \mathbb{N}^+_{\leq n}}) \leftarrow \pi_{\texttt{ref}}(\mathcal{G})$;

while $t > 0$ **do**

  for $i = 1, \ldots, n$ **do**

    for $s = 1, \ldots, b$ **do**

      $\tilde{\mathbf{R}}^i_{\theta,t}(f^i, s) = \mathbf{R}^i_t(s, f^i) \sum_{f^i_0} \frac{q_{t|0}(s|f^i_0)}{q_{t|0}(f^i|f^i_0)} p_\theta(f^i|\mathcal{G}_t, t)$;

      $J_{f^i,s} \leftarrow \text{Poisson}(\tau \mathbf{R}^i_t(s, f^i))$ `// # of transitions for a node`

    end

  end

  for $i, j = 1, \ldots, n$ **do**

    for $s = 1, \ldots, a$ **do**

      $\tilde{\mathbf{R}}^{(i,j)}_{\theta,t}(e^{(i,j)}, s) = \mathbf{R}^{(i,j)}_t(s, e^{(i,j)}) \sum_{e^{(i,j)}_0} \frac{q_{t|0}(s|e^{(i,j)}_0)}{q_{t|0}(e^{(i,j)}|e^{(i,j)}_0)} p_\theta(e^{(i,j)}|\mathcal{G}_t, t)$;

      $J_{e^{(i,j)},s} \leftarrow \text{Poisson}(\tau \mathbf{R}^{(i,j)}_t(s, e^{(i,j)}))$ `// # of transitions for an edge`

    end

  end

  for $i = 1, \ldots, n$ **do**

    if $\sum_{s=1}^b J_{f^i,s} > 1$ *or* $\sum_{s=1}^b J_{f^i,s} = 0$ **then**

      $f^i \leftarrow f^i$ `// stay the same`

    end

    else

      $s^* = \arg\max_s \{J_{f^i,s}\}_{s=1}^b$;

      $f^i \leftarrow s^*$ `// update node`

    end

  end

  for $i, j = 1, \ldots, n$ **do**

    if $\sum_{s=1}^a J_{e^{(i,j)},s} > 1$ *or* $\sum_{s=1}^a J_{e^{(i,j)},s} = 0$ **then**

      $e^{(i,j)} \leftarrow e^{(i,j)}$ `// stay the same`

    end

    else

      $s^* = \arg\max_s \{J_{e^{(i,j)},s}\}_{s=1}^b$ $e^{(i,j)} \leftarrow s^*$ `// update edge`

    end

  end

  $t \leftarrow t - \tau$;

end

feature.

The above node-level features and graph-level features are concatenated together as the auxiliary node-level features $\mathbf{F}_{\texttt{aux}}$ and graph-level features $\mathbf{y}$. An important property is that the above node-level features are permutation-equivariant, and the above graph-level

features are permutation-invariant, whose proof is straightforward, so we omit it here.

Next, two important modules used in the MPNN backbone, PNA and FiLM, are described in detail.

PNA Module. The PNA module [213] is implemented as follows,

$$\texttt{PNA}(\{\mathbf{x}_i\}_{i=1}^n) = \texttt{MLP}(\texttt{min}(\{\mathbf{x}_i\}_{i=1}^n) \oplus \texttt{max}(\{\mathbf{x}_i\}_{i=1}^n) \oplus \texttt{mean}(\{\mathbf{x}_i\}_{i=1}^n) \oplus \texttt{std}(\{\mathbf{x}_i\}_{i=1}^n)) \quad (4.98)$$

where $\oplus$ is the concatenation operator, $\mathbf{x}_i \in \mathbb{R}^d$; `min`, `max`, `mean`, and `std` are coordinate-wise, e.g., $\texttt{min}(\{\mathbf{x}_i\}_{i=1}^n) \in \mathbb{R}^d$.

FiLM Module. FiLM [214] is implemented as follows,

$$\texttt{FiLM}(\mathbf{x}_i, \mathbf{x}_j) = \texttt{Linear}(\mathbf{x}_i) + \texttt{Linear}(\mathbf{x}_i) \odot \mathbf{x}_j + \mathbf{x}_j \quad (4.99)$$

where `Linear` is a single fully-connected layer without activation function and $\odot$ is the Hadamard product.

4.2.12 Hyperparameter Settings

Hyperparameter Settings for Forward Diffusion. As we introduced in Proposition 4.5, we tried two sets of rate matrices for the node and edge forward diffusion, so that the converged distribution is either uniform or the marginal distribution. We found that the marginal distribution leads to better results than the uniform distribution. Thus, the reference distribution is the marginal distribution for all the main results, except Tables 4.14 and 4.16. The performance comparison between the marginal diffusion and uniform diffusion is presented in the ablation study in Sections 4.2.6 and4.2.6. The $\beta(t)$ controls how fast the forward process converges to the reference distribution, which is set as $\beta(t) = \alpha\gamma^t log(\gamma)$, which is consistent with many existing works [195, 196, 199]. In our implementation, we assume the converged time $T = 1$ and for the forward diffusion hyperparameters (α, γ) we tried two sets: $(1.0, 5.0)$ and $(0.8, 2.0)$ where the former one can ensure at $T = 1$ the distribution is very close to the reference distribution, and the latter one does not fully corrupt the raw data distribution so the graph-to-graph model $p^{\theta}_{0|t}$ is easier to train.

Hyperparameter Settings for Reverse Sampling. The number of sampling steps is determined by τ, which is $\texttt{round}(\frac{1}{\tau})$ if we set the converged time $T = 1$. We select the number of sampling steps from $\{50, 100, 500\}$, which is much smaller the number of sampling steps

of DiGress [197] from $\{500, 1000\}$. For the number of nodes n in every generated graph, we compute a graph size distribution of the training set by counting the number of graphs for different sizes (and normalize the counting to sum it up to 1). Then, we will sample the number of nodes from this graph size distribution for graph generation.

Hyperparameter Settings for Neural Networks. For DisCo-GT, the parametric graph-to-graph model $p^{\theta}_{0|t}$ is graph transformer (GT). We use the exactly same GT architecture as DiGress [197] and adopt their recommended configurations [8]. The reason is that this architecture is not our contribution, and setting the graph-to-graph model $p^{\theta}_{0|t}$ same can ensure a fair comparison between the discrete-time graph diffusion framework (from DiGress) and the continuous-time graph diffusion framework (from this work). For DisCo-MPNN, we search the number of MPNN layers from $\{3, 5, 8\}$, set all the hidden dimensions the same, and search it from $\{256, 512\}$. For both variants, the dropout is set as 0.1, the learning rate is set as $2e^{-4}$, and the weight decay is set as 0.

4.3 HOW TO MAKE LMS STRONG NODE CLASSIFIERS?

4.3.1 Introduction

There is a growing trend of utilizing Language Models (LMs) for machine learning tasks across diverse domains. This approach has shown tremendous promise in areas such as vision [236], audio [237], and multimodal learning [238]. In graph learning, recent efforts have begun to explore the capabilities of LMs in understanding and processing graph structures. [239] showed that LMs can detect node connectivity and identify cycles, while [240] explored LMs' ability to evaluate graph scale and identify connected components. Furthermore, InstructGLM [241] and LLaGA [242] achieved state-of-the-art (SOTA) performance in text-output node classifiers on Text-Attributed Graphs (TAG) [243], whose nodes have textual features.

However, both InstructGLM and LLaGA suffer from a fundamental limitation that compromises the generality of the backbone LM. Specifically, InstructGLM expands the LM's vocabulary by creating unique tokens for nodes, whose token embeddings are topology-aware node embeddings. It comes at the cost of incompatibility with two important use cases: (1) multi-task learning on diverse datasets, a common strategy for training Foundational Models [244, 245], and (2) certain personalized LM fine-tuning services [246] that

[8] `https://github.com/cvignac/DiGress/tree/main/conFigures/DisCo/experiment`

restrict modifications or access to the backbone model architecture/code[9]. LLaGA uses a shared text encoder and a projector to overcome the first limitation but still bears inflexibility when deploying different LMs and cannot be applied to LMs without code/architecture access. The above discussion raises a crucial question: *How can off-the-shelf,* ***text-to-text*** *instruction-tuned LMs [247] achieve competitive performance in node classification tasks without architectural modifications?*

In stark contrast to [248], which suggests that LMs may only interpret graph structures in prompts as contextual paragraphs, our work presents a more optimistic outlook. We aim to overcome this inherent limitation by augmenting the LMs' input while preserving their original architecture. Our proposed model, AuGLM (Aumented Graph Language Model), leverages two key augmentation strategies to enhance the LM's ability to process graph data:

- **Relevant Node Retrieval**: In contrast to InstructGLM, which relies on multi-hop ego networks akin to message-passing GNNs for structure-aware contextualization, AuGLM draws inspiration from Graph Transformers (GTs) [201, 249] and Retrieval-Augmented Generation (RAG) [250, 251]. This enables the LM to access long-range structural and semantic information about the target node. We propose two complementary approaches to achieve this: (1) topological retrieval, and (2) prototypical semantic retrieval.

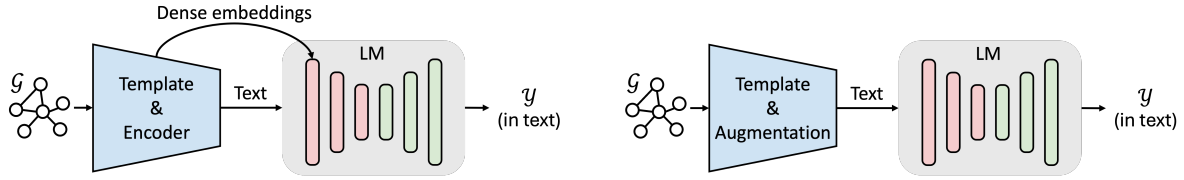
- **Candidate Label Pruning**: To improve LMs' understanding of graph data while maintaining their text-to-text architecture, we convey the guidance from a specialist model, a pretrained lightweight GNN, to the input of LMs via narrowing down the candidate labels. This allows LMs to focus on discerning between closely related candidates, ultimately enhancing the performance.

We extensively evaluate our approach on four real-world TAGs, showing the effectiveness of AuGLM. The results indicate that backbone LMs augmented with AuGLM consistently outperform SOTA text-output classifiers while also matching or surpassing the performance of SOTA vector-output classifiers. These findings represent a crucial step towards bridging the gap between tailored task-specific node classifiers and more general, fine-tuned LMs, highlighting the potential for unified models excelling in multiple tasks.

4.3.2 Preliminaries

We use the following notation conventions: bold lower-case letters (e.g,. $\mathbf{x}$) denote column vectors, bold upper-case letters (e.g., $\mathbf{X}$) denote matrices, and calligraphic upper-case letters

[9] `https://platform.openai.com/docs/guides/fine-tuning`



(a) InstructGLM and LLaGA. (b) AuGLM (ours).

Figure 4.10: Comparison of pipelines between the existing LM-based node classifiers and our approach, AuGLM. Unlike InstructGLM and LLaGA, which explicitly encodes graph information into token embeddings as a form of soft prompting, AuGLM maintains the original text-to-text framework of the off-the-shelf LM, offering greater generality and flexibility.

(e.g., $\mathcal{X}$) denote sets. We use $[\cdot]$ and $[\cdot, \cdot]$ to index vectors and matrices, respectively.

We study the node classification problem on TAGs where each node is associated with textual attributes. A TAG with n nodes is represented as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T})$, where $\mathcal{V} = \{v_i\}_{i=1}^n$ denotes a set of nodes, and $\mathcal{E} = \{e_{ij}\}_{i,j=1}^n$ is a set of edges where $e_{ij} = 1$ indicates that nodes v_i and v_j are connected; otherwise, $e_{ij} = 0$. $\mathcal{T} = \{t_i\}_{i=1}^n$ indicates the set of node textual attributes. The edges can also be represented by an adjacency matrix $\mathbf{A} \in \{0, 1\}^{n \times n}$, where $\mathbf{A}[i, j] = 1$ if and only if $e_{ij} = 1$. The training and test node labels are denoted by $\mathcal{Y} = \mathcal{Y}_{\text{train}} \cup \mathcal{Y}_{\text{test}} = \{y_i\}_{i=1}^n$, where each label y_i belongs to one of the C classes, i.e., $y_i \in \{1, \ldots, C\}, \forall i$. In the semi-supervised setting studied in this paper, the graph structure and training labels $\mathcal{V}, \mathcal{E}, \mathcal{T}, \mathcal{Y}_{\text{train}}$ are accessible during training. The task is to predict the labels of test nodes $\mathcal{Y}_{\text{test}}$.

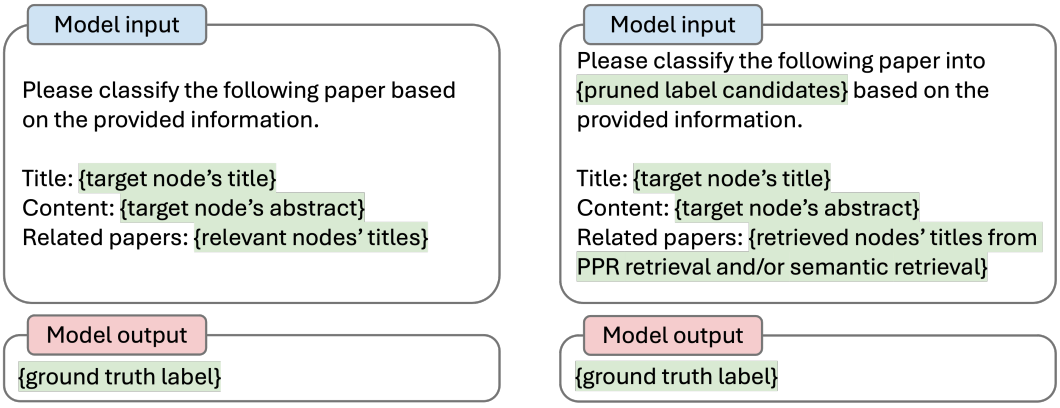
Personalized PageRank (PPR). PPR [12, 252] ranks all the nodes according to their relevance to a given query node. Specifically, given the adjacency matrix $\mathbf{A}$, the PPR scores $\mathbf{r}_i \in \mathbb{R}^n$ for all nodes concerning the query node v_i are computed iteratively as:

$$\mathbf{r}_i \leftarrow (1 - \alpha)\tilde{\mathbf{A}}\mathbf{r}_i + \alpha\mathbf{q}_i \tag{4.100}$$

where $\alpha \in (0, 1)$ is the teleport probability, $\mathbf{q}_i \in \{0, 1\}^n$ is a one-hot vector whose i-th entry is 1, $\tilde{\mathbf{A}} = \mathbf{A}\mathbf{D}^{-1}$ is the normalized adjacency matrix, and $\mathbf{D}$ is the degree matrix. Once $\mathbf{r}_i$ converges, the top-K relevant nodes concerning the query node v_i can be identified as follows:

$$\texttt{PPR}(v_i, K) = \{v_j : \mathbf{r}_i[j] \in \text{topK}(\mathbf{r}_i)\} \tag{4.101}$$

Language Models (LMs). We employ autoregressive LMs that predict the next token z_i based on the input sequence t and the context of previously generated tokens $z_{1:i-1}$. The



(a) A typical graph-to-text template. (b) Our template with augmented text.

Figure 4.11: Comparison of a typical graph-to-text template (a) and our template with augmented text features (b).

probability of generating a sequence z given the input t is:

$$p_{\text{LM}}(z|t) = \prod_{i=1}^{|z|} p_{\text{LM}}(z_i|t, z_{1:i-1}) \tag{4.102}$$

Retrieval-Augmented Generation (RAG). RAG [250, 251] first retrieves a query t-relevant text d^* from an external corpus $\mathcal{D}$ via a similarity function s_ϕ:

$$d^* = \arg\max_{d \in \mathcal{D}} s_\phi(d, t) \tag{4.103}$$

s_ϕ is typically implemented as a dual-encoder architecture [253]:

$$s_\phi(d, t) = \langle \texttt{Encoder}_\phi(d), \texttt{Encoder}_\phi(t) \rangle \tag{4.104}$$

Once d^* is retrieved, it is fed into the LM together with the initial query t: $p_{\text{LM}}(z|d^*, t)$ for generation.

4.3.3 Method

We explore the application of LMs to node classification, leveraging the instruction tuning to reformulate node classification as a text-to-text task [247]. Our method employs carefully designed prompt templates and augmentation techniques to transform graph data and ground truth labels into text pairs, enabling LMs to process and be fine-tuned without modifying their underlying architecture.

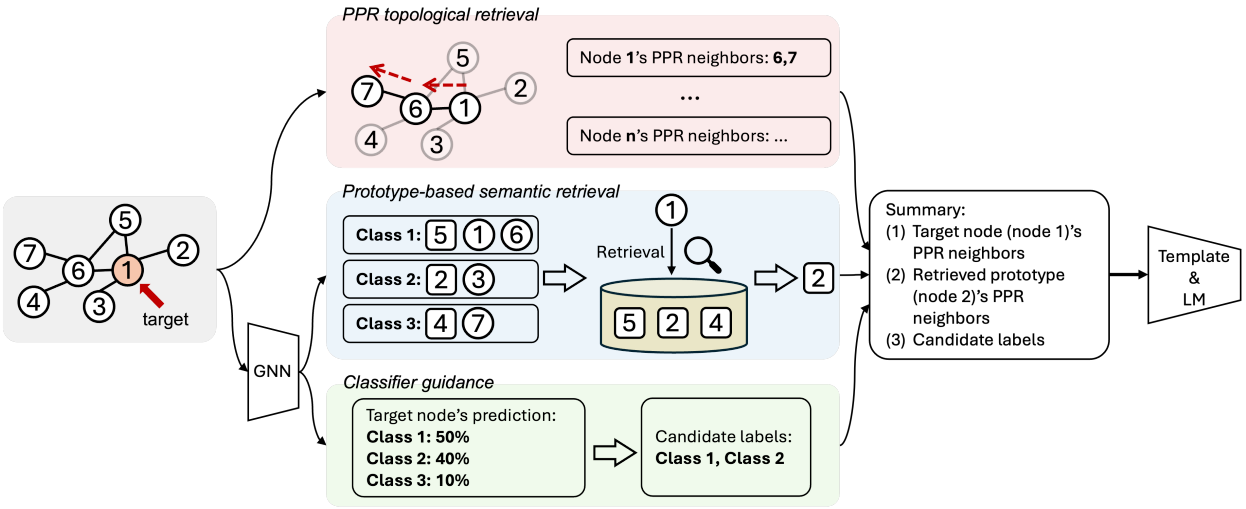


Figure 4.12: A detailed pipeline of AuGLM. In the semantic retrieval module, rectangle nodes denote the class prototypes.

As shown in Figure 4.10, AuGLM fundamentally differs from InstructGLM [241] and LLaGA [242], the current SOTA LM node classifiers. While all three methods utilize prompt templates to transform input graphs into text, InstructGLM and LLaGA explicitly **encode node features into the LM's token embeddings** which can be categorized as **soft prompting** [254]. In contrast, our approach provides a **data augmentation**-based framework without modifying the LM's text-to-text architecture, enabling our model to retain the versatility of the original LM. The following section first details the augmentation techniques developed by this paper and then introduces the templates to incorporate all the augmented textual features.

Retrieval-based Aggregation. General LMs are not designed to process graph data directly. To overcome this, a common approach is to employ prompt templates to transform graph data and associated tasks into text that LMs can understand. For instance, for the Cora [255] literature citation graph, a typical template [241, 248] for node classification, as shown in Figure 4.11a, consists of three main components: (1) a short classification task description, (2) the target node's textual features, e.g., its title and abstract, and (3) textual features from relevant nodes.

The success of the message-passing GNNs highlights the importance of the **aggregation** operation, whose typical example is the mean pooling of intermediate node embeddings. A similar spirit is followed for the LM-based classifiers, whose key design is the **selection of relevant nodes**. Existing works [241, 248] select one/multi-hop neighbors as relevant nodes, but we posit that this approach is suboptimal for two reasons. Firstly, not all immediate or

extended neighbors are relevant to the target node, which can introduce noise and degrade model performance. Secondly, incorporating multi-hop neighbors can lead to "neighbor explosion" [256, 257, 258], i.e., an exponentially-growing set of "relevant" nodes, resulting in increased computational costs, slower model training/inference, and even lead to the out-of-memory issue. In response, two novel solutions, *topological retrieval* and *prototypical semantic retrieval*, are proposed to identify the most informative nodes for classification tasks efficiently.

Topological Retrieval. PPR [12, 252] is leveraged for topological retrieval, which has shown great effectiveness in conjunction with GNNs [259]. The success of PPR suggests that its retrieved neighbors may provide more informative context than generic one-hop or multi-hop neighbors. Specifically, for a target node v_i, we select its top-K neighbors $\texttt{PPR}(v_i, K)$ based on their PPR scores (Eqs. (4.100) and (4.101), Section 4.3.2). Then, the text features from the PPR neighbors are concatenated as the PPR-retrieved text $t_{\text{PPR}} = \oplus_{j;v_j \in \texttt{PPR}(v_i,K)} t_j$, where $\oplus$ denotes text concatenation.

It is worth noting that the classic PPR algorithm is computationally expensive for large graphs due to the matrix multiplication (Eq. (4.100)). However, efficient approximate solutions such as ApproximatePR [260], can be applied to mitigate this issue. Nevertheless, PPR is a topology-based heuristic that inherently cannot leverage textual features or supervision from downstream LMs. To enhance our framework's semantic awareness, we propose a complementary strategy, called prototypical semantic retrieval, which is discussed below.

Prototypical Semantic Retrieval. Our semantic retrieval module draws inspiration from two popular techniques: (1) RAG [250, 251], which retrieves external corpora, and (2) Graph Transformers [201], which aggregate messages from distant nodes via inner product-based attention weights. In the context of node classification, we treat **the textual features of all nodes except the target node** as a surrogate "external corpus." However, unlike typical question-answering tasks [261, 262], retrieving textual features from **a single node is often insufficient** for accurate node classification. To address this, we enhance the semantic retrieval by retrieving prototypes, which capture the essence of each class [131].

We employ prototypes [263] as representative examples in the classification problem. To obtain these prototypes, a lightweight GNN ψ is pretrained in a semi-supervised way and generates a prediction vector for each node: $\tilde{\mathbf{y}}_i = \texttt{GNN}_\psi(v_i, \mathcal{G}) \in \mathbb{R}^C, \forall v_i$. The prediction confidence for each node v_i is defined as: $\texttt{Conf}(v_i) = \max_j \tilde{\mathbf{y}}_i[j]$. The predicted class-$c$ examples are $\tilde{\mathcal{Y}}_c = \{v_i : \arg\max_j \tilde{\mathbf{y}}_i[j] = c\}$ and their confidence is $\texttt{Conf}_c = \{\texttt{Conf}(v_i) : v_i \in$

$\tilde{\mathcal{Y}}_c\}$. For each class c, the top-N confident examples are selected as prototypes:

$$\mathcal{P}_c = \left\{ v_i : v_i \in \tilde{\mathcal{Y}}_c \wedge \texttt{Conf}(v_i) \in \text{topN}\left(\texttt{Conf}_c\right) \right\} \tag{4.105}$$

For all the classes, there are $N \times C$ prototypes: $\mathcal{P} = \bigcup_{c\in\{1,\ldots,C\}} \mathcal{P}_c$. To ensure every document in the corpus $\mathcal{D}$ includes text features from **multiple nodes**, $\mathcal{D}$ is constructed by concatenating the text of **each prototype's PPR neighbors**:

$$\mathcal{D} = \left\{ \bigoplus_{j;v_j\in\texttt{PPR}(v_i,K)} t_j : v_i \in \mathcal{P} \right\} \tag{4.106}$$

Next, for each target node with its associated text features t_{target}, we compute the prototypically retrieved text using Eq. (4.103): $t_{\text{proto}} = \arg\max_{d\in\mathcal{D}} s_\phi(d, t_{\text{target}})$. In our experiments, we may use t_{PPR} (from topological retrieval), or t_{proto}, or both by concatenation $t_{\text{PPR}} \oplus t_{\text{proto}}$. For simplicity, we denote the final retrieved text as t_{retri}.

Classifier Guidance. Recent studies [240, 248, 264] highlighted mainstream LMs' limited understanding of graph topology. While InstructGLM [241] and LLaGA [242] address this limitation by incorporating topology-aware node embeddings (e.g., from a pretrained or parameter-free GNN) into the LM's token embeddings, this approach necessitates **modifications to the LM's architecture**. We propose an alternative method that conveys guidance from a pretrained GNN into the **input text of LMs**, thereby preserving the LM's original architecture. Concretely, such guidance is to prune the classification candidates.

We repurpose the pretrained $\texttt{GNN}_\psi$ from the prototypical semantic retrieval module. For each node v_i, we identify and save the top-I predicted labels:

$$\mathcal{L}_i = \{j : \tilde{\mathbf{y}}_i[j] \in \text{topI}\left(\tilde{\mathbf{y}}_i\right)\} \in \{1,\ldots,C\}^I \tag{4.107}$$

where $I < C$. For datasets in the experiments, the `IndexToLabel` maps are available, which map numerical labels to their corresponding text. The pruned label candidates for node v_i can be presented as concatenated text: $t_{\text{candidates}} = \oplus_{i\in\mathcal{L}_i} \texttt{IndexToLabel}(i)$. The integration of pruned candidates into the template is detailed in Section 4.3.3.

By focusing on a more relevant set of candidate labels, valuable topology-aware inductive bias from the GNN is incorporated into the LM's input, thereby enhancing its ability to perform node classification without altering its fundamental architecture.

Overall Template. Our augmented training samples include three key elements: (1) the target node's text t_{target}, (2) the retrieved nodes' text t_{retri}, and (3) the pruned label candidates $t_{\text{candidates}}$. We collectively denote these elements as $t_{\text{in}} = (t_{\text{target}}, t_{\text{retri}}, t_{\text{candidates}})$. LM's prediction probability for the target sequence y_{target} is based on Eq. (4.102) whose input text t is t_{in} and the output sequence z is y_{target}.

Figure 4.11b presents an exemplar template for the Cora dataset, showcasing the integration of t_{target}, t_{retri}, and $t_{\text{candidates}}$. The selection of the backbone LM will be detailed in Section 4.3.4. Section 4.3.7 contains a complete list of templates. Note that we exclude the "abstracts" of the retrieved nodes to prevent exceeding the maximum input length of most LMs. We utilize only this template's "model input" part during evaluation.

Training. Our framework includes three parameterized modules that require training or fine-tuning: (1) GNNs for generating prototypes and candidate label pruning, as described in Sections 4.3.3 and 4.3.3, (2) the encoder ϕ from the semantic retriever, defined in Eq. 4.104, and (3) the backbone LM, utilized in Eq. 4.102. The GNNs from Sections 4.3.3 and 4.3.3 can be shared, and their training is independent of the other modules, which are supervised by ground truth labels. This process is detailed in Section 4.3.5.

One of the standard LM's losses, the average token-wise negative log-likelihood (NLL), is used. For a target node, the loss is:

$$\mathcal{L}_{\text{NLL}}(p_{\text{LM}}(y_{\text{target}}|t_{\text{in}}), y_{\text{target}}) \tag{4.108}$$

To train the semantic retriever, we employ a distribution-matching loss. For a given target node's text t_{target}, its retrieval probability for a prototype text $t \in \mathcal{D}$ is:

$$p_\phi(t|t_{\text{target}}) = \frac{e^{s_\phi(t,t_{\text{target}})}}{\sum_{t' \in \mathcal{D}} e^{s_\phi(t',t_{\text{target}})}} \tag{4.109}$$

Next, an empirical distribution supervised by the LM is:

$$\tilde{p}_{\text{LM}}(t|t_{\text{target}}, y_{\text{target}}) = \frac{e^{p_{\text{LM}}(y_{\text{target}}|t_{\text{in}})}}{\sum_{t' \in \mathcal{D}} e^{p_{\text{LM}}(y_{\text{target}}|t'_{\text{in}})}} \tag{4.110}$$

where $t_{\text{in}} = (t_{\text{target}}, t, t_{\text{candidates}})$ and $t'_{\text{in}} = (t_{\text{target}}, t', t_{\text{candidates}})$. This distribution represents the **normalized importance of each prototype text** $t \in \mathcal{D}$ based on the **LM's likelihood of generating the ground truth text** y_{target}. We use $\tilde{p}_{\text{LM}}$ to distinguish this distribution from the generation probability in Eqs. (4.102).

The distribution matching loss is the Kullback-Leibler (KL) divergence between the re-

Algorithm 4.2: Training AuGLM

Given: (1) A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T})$ and training labels $\mathcal{Y}_{\text{train}}$, (2) initialized backbone LM θ, (3) initialized semantic encoder ϕ, and (4) initialized GNN ψ;
Preprocessing: (1) train the GNN ψ based on $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T})$ and $\mathcal{Y}_{\text{train}}$ till convergence; (2) generate prototypes and their text via Eqs. (4.105) and (4.106); (3) generate the pruned label candidates for every node via Eq. (4.107).;
while *θ and ϕ not converged* **do**
    Sample $v_i \sim \mathcal{V}$ whose text is t_i.;
    Retrieve the relevant nodes' text $t_{\text{retri},i}$ via topological (Eq. (4.101)) and/or semantic retrieval (Eq. (4.103)).;
    Plug t_i, $t_{\text{retri},i}$, and $t_{\text{candidates},i}$ (from preprocessing (3)) into the template (e.g., Figure 4.11b), compute the NLL loss by Eq. (4.108), and update θ.;
    Compute retrieval distribution $p_\phi(\cdot|t_i)$ by Eq. (4.109).;
    Call LM inference $|\mathcal{D}|$ times for $\{p_{\text{LM}}(y_i|t_i, t)\}_{t\in\mathcal{D}}$ and $\tilde{p}_{\text{LM}}(\cdot|t_i, y_i)$.;
    Compute retriever loss by Eq. (4.111) and update ϕ.;
end

trieved and the LM-supervised distributions:

$$\text{KL}\left(\texttt{sg}\left(\tilde{p}_{\text{LM}}\left(\cdot|t_{\text{target}}, y_{\text{target}}\right)\right) \| p_\phi(\cdot|t_{\text{target}})\right) \tag{4.111}$$

This loss aims to align the retrieved probability of each prototype text $t \in \mathcal{D}$ with its importance in facilitating the LM's generation of the label text y_{target} for the target node. The stop gradient operator `sg` ensures that only the semantic retriever ϕ is updated while keeping the LM's parameters θ frozen. This objective has been used by previous works [265, 266] without a thorough analysis. We examine its properties and implications in Section 4.3.6.

Notably, computing Eq. (4.110) requires $|\mathcal{D}|$ inferences of the LM due to the denominator. However, the LM is fine-tuned only on the NLL loss for the most relevant prototype, $\arg\max_{d\in\mathcal{D}} s_\phi(d, t_{\text{target}})$ via Eq. (4.108). Consequently, each update step involves $|\mathcal{D}|$ forward passes but only one backpropagation. To further reduce the computational overhead associated with $|\mathcal{D}|$ inferences, we can employ a typical sampling strategy: selecting the top-M samples to form a retrieval minibatch $\mathcal{D}_M = \{t : t \in \text{topM}_{t'\in\mathcal{D}} s_\phi(t', t_{\text{target}})\}$. By replacing $\mathcal{D}$ with $\mathcal{D}_M$ in Eqs. (4.109) and (4.110), the retrieval and the LM-supervised distributions can be computed "in-batch", reducing the total inference times from $|\mathcal{D}|$ to M.

Algorithm 4.2 outlines a step-by-step process for fine-tuning AuGLM, processing one training node per step. This procedure can be readily extended to mini-batch settings.

Table 4.17: Accuracy (%) comparison between AUGLM and existing SOTA models. The best-performing **vector-output** and **text-output** models on each dataset are highlighted in **blue** and **red**, respectively.

	Method	Cora	Pubmed	ogbn-arxiv	ogbn-products
Vector-output	GCN	87.78±0.96	88.90±0.32	73.60±0.18	75.64±0.21
	GraphSAGE	86.51±2.36	89.08±0.28	73.88±0.33	76.04±0.25
	BernNet	88.52±0.95	88.48±0.41	–	–
	FAGCN	88.85±1.36	89.98±0.52	–	–
	GCNII	88.98±1.33	89.80±0.52	72.74±0.16	–
	ACM-GCN	89.75±1.16	91.44±0.59	–	–
	GLEM + RevGAT	88.56±0.60	94.71±0.20	76.97±0.19	–
	GIANT + RevGAT	83.53±0.38	85.02±0.48	75.90±0.19	71.89±0.30
	GIANT + GCN	84.23±0.53	84.19±0.50	73.29±0.10	69.77±0.42
	DeBERTa	76.06±3.78	94.94±0.46	73.61±0.04	72.97±0.23
	TAPE + RevGAT	**92.90±3.07**	**96.18±0.53**	**77.50±0.12**	**82.34±0.36**
Text-output	ChatGPT-3.5	67.90	93.42	73.40	74.40
	InstructGLM	90.77±0.52	94.62±0.13	75.70±0.12	–
	LLaGA	89.85	95.06	76.66	–
	AUGLM (T5-small)	91.14±0.55	94.80±0.15	75.39±0.21	81.73±0.08
	AUGLM (T5-base)	91.24±0.46	95.03±0.35	**76.80±0.14**	81.91±0.11
	AUGLM (T5-large)	**91.51±0.26**	**95.16±0.18**	76.00±0.23	**82.90±0.10**

Model Complexity. AUGLM consists of three parameterized modules: (1) a GNN ψ, (2) the semantic retriever ϕ, and (3) the backbone LM θ. Notably, ψ and ϕ are lightweight, with the number of parameters being only 1/30 to 1/3 of the number of LM θ parameters. Compared to the SOTA InstructGLM [241], AUGLM has an additional module ϕ, resulting in slightly more parameters, which are relatively minor. For training, the GNN ψ can be trained independently, and the PPR scores can be precomputed. The training of θ relies on the retrieved text from ϕ, while the training of ϕ requires $\tilde{p}_{\text{LM}}(\cdot|t_{\text{target}}, y_{\text{target}})$, which is obtained through forward inference of θ. Importantly, computational graphs (used for gradient computation) of θ and ϕ are **independent**. When training ϕ, the stop gradient operator `sg` ensures θ has no gradient. As a result, the cost of backpropagation is similar to updating the LM θ and the semantic encoder ϕ separately.

4.3.4 Experiments

Setups. Following [241, 267], we evaluate our approach on four benchmark datasets: Cora [255], Pubmed [255], ogbn-arxiv [268], and a subset of ogbn-products [267, 268]. The dataset statistics are in Table 4.18.

Our implementation employs two pretrained all-MiniLM-L6-v2 models [269] as the the semantic retriever ϕ (Eq. (4.104)) and the text encoder for GNN ψ (Eq. (4.112)). We set the PPR teleport probability $\alpha = 0.1$. We employ a 3-layer GraphSAGE [256] with a hidden dimension of 256 as ψ. Our hyperparameter settings include $K = 5$ PPR neighbors, $N = 10$ prototypes, and $M = 8$ samples for LM inference. The number of label candidates I is searched from $\{2, 3\}$. Flan-T5-small/base/large [270] are used as the backbone LM θ, whose parameters are instruction-tuned using the templates in Section 4.3.7.

Dataset Statistics. We present the detailed statistics of datasets used in this paper in Table 4.18.

All the baseline methods' performance on the Cora, Pubmed, and ogbn-arxiv is reported from the public leaderboards [10][11][12] and their published papers.

The ogbn-products dataset used in this paper is a subset of the original ogbn-products dataset [268] from TAPE [267]. We follow the settings in TAPE and report baseline methods' performance from the TAPE [267] paper.

Table 4.18: Dataset statistics.

Name	# Nodes	# Edges	# Classes	Split Strategy	Evaluation Metric
Cora	2 708	10 556	7	Random 60/20/20%	Accuracy
Pubmed	19 717	88 648	3	Random 60/20/20%	Accuracy
ogbn-arxiv	169 343	1 166 243	40	Given split	Accuracy
ogbn-products	54 025	198 663	47	Given split	Accuracy

Comparison with State of the Arts. This section presents the comparison between AuGLM and SOTA baselines. We categorize models into two groups: (1) vector-output models, which output a vector with dimension equal to the number of classes, and (2) text-output models, whose output is text. Specifically, results from GCN [271], BernNet [272], FAGCN [273], GCNII [274], ACM-GCN [275], GLEM [276]+RevGAT, InstructGLM [241] and LLaGA [242] are reported according to the leaderboards and their papers. The results for TAPE+RevGAT, GIANT [277]+RevGAT [278], GIANT+GCN, DeBERTa [279], and ChatGPT3.5 are reported from [267]. All models are (at least partially) fine-tuned on the training set except ChatGPT-3.5. Mean and standard deviation over 5 runs are reported.

[10] `https://paperswithcode.com/sota/node-classification-on-cora-60-20-20-random`
[11] `https://paperswithcode.com/sota/node-classification-on-pubmed-60-20-20-random`
[12] `https://ogb.stanford.edu/docs/leader_nodeprop/`

Table 4.19: FLOPs comparison between different modules.

Module	**FLOPs** (10^9)
Retriever	2.3
T5-small	71.7
T5-base	257.2
T5-large	845.4

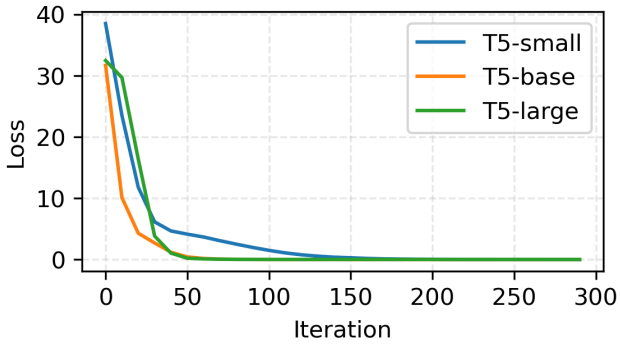


Figure 4.13: Convergence curve of AuGLM.

For text-output models, accuracy is evaluated by checking whether the model's generated text matches the ground truth text exactly.

Table 4.17 presents a comparison between AuGLM and SOTAs. AuGLM consistently outperforms InstructGLM and LLaGA, **achieving new SOTA performance** among text-output node classifiers. Notably, this superior performance is achieved without modifying any of the LMs' architectures, demonstrating the effectiveness of our approach. Furthermore, AuGLM exhibits **competitive performance compared to the best vector-output models**. Specifically, on Cora, Pubmed, and ogbn-arxiv datasets, AuGLM performs closely to that of the SOTA vector-output models. Furthermore, on the ogbn-products dataset, AuGLM surpasses the performance of the best vector-output model, TAPE.

Efficiency Study: FLOPs. The floating point operations (FLOPs) of AuGLM are studied. Specifically, the computation of our AugGLM includes (1) precomputing PPR neighbors for every node, (2) training and inference of the semantic retriever ϕ, and (3) training and inference of the LM θ. Hence, the extra on-the-fly computation cost is from the semantic retriever ϕ (all-MiniLM-L6-v2 in our experiments). We report the FLOPs of the retriever and different LM backbones in Table 4.19. The results show that (1) the retriever only adds a tiny amount of FLOPs to the backbone LMs and (2) our proposed AuGLM is efficient.

Table 4.20: Running time (ms) of different modules.

Module	Forward	Backprop
Retriever	14.7	6.1
T5-small	90.0	32.0
T5-base	104.4	66.6
T5-large	277.2	197.0

Efficiency Study: Convergence Curve. We train AuGLM with different backbone LMs: FLAN-T5-small/base/large on the Cora dataset and plot their loss curves regarding updating steps in Figure 4.13. In this experiment, the batch size is 16. It shows that our proposed AuGLM converges smoothly and quickly when equipped with various LMs of different scales.

Efficiency Study: Running Time. The running time (both forward and backpropagation) of the semantic retriever and the backbone LMs on the Cora dataset is recorded. The batch size is 1. This experiment is tested on an NVIDIA A100-SXM4-80GB. Table 4.20 shows that the semantic retriever only adds very limited on-the-fly computation overhead compared to the LM, demonstrating the efficiency of AuGLM.

Efficiency Study: Memory Usage. Memory usage is linear with respect to batch size. We report the memory usage of AuGLM with different backbone LMs in Table 4.21, where we set the batch size to 1 and we found the experimental results reasonable because more powerful backbone LMs require more GPU memory.

Table 4.21: GPU Memory usage (MB) with different LMs.

Model	Memory
AuGLM (T5-small)	3 098
AuGLM (T5-base)	6 572
AuGLM (T5-large)	20 308

Ablation Study. To evaluate the contribution of each key component in AuGLM, we conducted an ablation study on three crucial modules: (1) topological retrieval, (2) semantic retrieval, and (3) candidate label pruning. In this subsection, Flan-T5-small is used. The results in Table 4.22 demonstrate that each module consistently improves performance

Table 4.22: Ablation study results (accuracy %). T, S, and L denote topological retrieval, semantic retrieval, and label pruning, respectively. ↓ indicates accuracy drop compared to the full model (T+S+L).

Model	Cora	Pubmed	ogbn-arxiv	ogbn-products
T+S	85.52 (↓5.62)	94.40 (↓0.40)	72.91 (↓2.48)	79.83 (↓1.90)
T+L	87.27 (↓3.87)	94.32 (↓0.48)	73.79 (↓1.60)	81.05 (↓0.68)
S+L	90.25 (↓0.89)	94.26 (↓0.54)	73.46 (↓1.93)	79.06 (↓2.67)
T+S+L	**91.14**	**94.80**	**75.39**	**81.73**

Table 4.23: Joint vs. separate training (accuracy %).

Training	Cora	Pubmed	ogbn-arxiv	ogbn-products
Joint	91.52	94.52	74.87	82.29
Separate	91.14	94.80	75.39	81.73

across all datasets. Notably, our analysis reveals that the relative importance of each component varies across different datasets. For instance, candidate label pruning greatly impacts performance for the Cora dataset, whereas its effect is less pronounced for the ogbn-products dataset. This variation in component importance underscores the adaptability of our approach, which can effectively accommodate diverse datasets with different characteristics.

Multi-Task Training. One of the key advantages of pure text-to-text instruction tuning is that a single model can be trained on multiple tasks with the same input-output format. To verify this, AuGLM with Flan-T5-small is jointly trained on diverse datasets: Cora, Pubmed, ogbn-arxiv, and ogbn-products. The results in Table 4.23 show that the jointly trained model achieves performance comparable to models trained separately on each dataset. We observe that on specific datasets, such as Cora and ogbn-products, the jointly trained model outperforms its dataset-specific counterparts.

These findings suggest that our approach can effectively **handle multiple graph datasets using a single model**, without incurring significant performance losses compared to models trained individually. This capability is crucial for efficient model deployment when dealing with diverse graphs. In contrast, other approaches, such as InstructGLM, require the addition of a large token dictionary to accommodate all nodes in the joint dataset, which hinders their ability to achieve similar generality. Moreover, most vector-output models, including TAPE, are limited by their predefined input-output dimensions, making them inflexible and unable to handle multiple datasets.

Hyperparameter Study: Backbone GNNs. Specifically, we study the performance of AuGLM equipped with different GNNs. We compared the performance of AuGLM equipped with GraphSAGE (used in the reported results) with the counterpart equipped with GCN [271]. The comparison is in Table 4.24.

Table 4.24: Performance (accuracy %) comparison of AuGLM equipped with different GNNs.

Model	Cora	Pubmed	ogbn-arxiv	ogbn-products
GraphSAGE	91.14	94.80	75.39	81.73
GCN	90.98	94.85	75.21	81.82

We observed that the performance is nearly identical between GCN and GraphSAGE. This can be attributed to two factors: (1) the classification performances of GCN and GraphSAGE are similar, and (2) the GNN is used to generate prototypes and prune candidate labels, which **does not require a highly powerful GNN** for accurate classification.

Hyperparameter Study: Number of PPR Retrieved Nodes. Next, we examined the relationship between the model performance and the number of nodes retrieved. In this auxiliary experiment, we fixed the number of nodes retrieved by semantic retrieval at 5 and varied the number of nodes retrieved by PPR retrieval. The results are reported in Table 4.25

Table 4.25: Accuracy (%) of AuGLM on ogbn-arxiv with different numbers of PPR-retrieved neighbors. The best result is **bolded**.

# Neighbors	Accuracy (%)
1	75.18
3	75.76
5	75.39
7	75.19
9	76.05
10	**76.45**
15	75.99
20	74.81
25	74.48

Interestingly, we found that the model's performance remains relatively stable when the number of PPR nodes is less than 15. However, the performance degrades when too many nodes are retrieved (more than 15). A possible explanation is that when the number of PPR

Table 4.26: Accuracy (%) of AuGLM with different *topological* retrieval techniques across datasets. The best result for each dataset is **bolded**.

Retrieval Technique	Cora	Pubmed	ogbn-arxiv	ogbn-products
1-hop neighbors	90.59	94.33	73.97	79.53
GAE	90.83	94.42	74.01	79.85
PPR neighbors	**91.14**	**94.80**	**75.39**	**81.73**

nodes becomes too large, every target node's **retrieved nodes become similar** (e.g., some hub nodes are retrieved by most nodes), reducing the discriminativeness of each target node. This phenomenon is reminiscent of the "oversmoothing" problem [280] in GNNs, where a GNN with too many layers and too large receptive field produces indistinguishable latent representations for all nodes.

Hyperparameter Study: Other Topological Retrieval Options. In this auxiliary experiment, we use the link predictor to retrieve relevant neighbors. Specifically, we trained a graph autoencoder (GAE) [281], a basic graph neural network-based link predictor, on the given graph. Then, we retrieved the **top-**5 **most confident neighbors from the reconstructed graph** to replace those obtained through PPR retrieval. The results are presented in Table 4.26, where Flan-T5-small is used as the backbone LM. For better reference, we also provide a version where PPR retrieval is replaced with retrieving from 1-hop neighbors.

We observe that both 1-hop neighbor retrieval and GAE perform worse than their PPR counterparts. A possible reason is that both 1-hop neighbor retrieval and GAE are **local** retrieval methods, whereas PPR can effectively capture the **global** structure. Additionally, we note that GAE is trained using a reconstruction loss, which means it tends to assign high confidence to **existing edges**. In other words, the neighbors retrieved by GAE would be similar to those obtained through 1-hop neighbor retrieval, except for some low-degree nodes.

Hyperparameter Study: Other Semantic Retrieval Options. This additional experiment uses different semantic retrievers to replace the prototype-based semantic retriever used in the proposed AuGLM. In detail, the prototype-based semantic retrieval module is replaced with a simple semantic retriever that **selects the most textually similar nodes** via inner product. Concretely, we use two pretrained models, (1) original all-MiniLM-L6-v2[13]

[13] `https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2`

Table 4.27: Accuracy (%) of AuGLM with different *semantic* retrieval techniques across datasets. The best result for each dataset is **bolded**.

Retrieval Technique	Cora	Pubmed	ogbn-arxiv	ogbn-products
Simple semantic retriever	90.68	94.37	74.46	81.21
SimTeG-tuned simple retriever	—	—	74.70	—
Prototype-based retriever (ours)	**91.14**	**94.80**	**75.39**	**81.73**

and (2) fine-tuned all-MiniLM-L6-v2 by SimTeG [282][14]. The remaining modules, including topological retrieval and classifier guidance, were left intact, and FLAN-T5-small is used as the LM backbone. The results are reported below.

We observe that the proposed prototype-based retriever is better than both the original all-MiniLM-L6-v2-based retriever and the SimTeg-tuned simple retriever. This is because:

1. The training objective of the SimTeG-tuned retriever is to align the classification loss with a GNN model [282], similar to knowledge distillation [283]. In other words, **the SimTeG-tuned retriever is a mixture of topological and semantic retrieval**, as the GNN incorporates both topology and node features. This means that its role partially overlaps with that of the topological PPR retriever.

2. Our prototype-based retriever can retrieve textual features from **multiple nodes**, but the other two cannot achieve this.

Selected Hyperparameters. We report the hyperparameter used for every dataset in Table 4.28. As mentioned in the main content, we use two pretrained all-MiniLM-L6-v2 models as the dual encoder and the Flan-T5-small/base/large models as the backbone; they are all publicly available[15][16]. More detailed hyperparameters will be released with the code upon publication.

4.3.5 Architecture and Training of the Graph Neural Network ψ

In our setting, semi-supervised node classification problem, $\mathcal{V}, \mathcal{E}, \mathcal{T}, \mathcal{Y}_{\text{train}}$ are available during training. Since Graph Neural Networks (GNNs) are not inherently capable of processing textual features, a pretrained text encoder is used to generate d-dimensional dense

[14] https://huggingface.co/datasets/vermouthdky/SimTeG/tree/main/ogbn-arxiv/all-MiniLM-L6-v2/main/cached_embs

[15] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

[16] https://huggingface.co/docs/transformers/en/model_doc/flan-t5

Table 4.28: Selected hyperparameters for AUGLM across different datasets.

Hyperparameter	**Cora**	**Pubmed**	**ogbn-arxiv**	**ogbn-products**
# PPR neighbors	5	2	5	5
# Semantic neighbors	5	2	5	5
Prompt template	Citation	Citation	Citation, Title Last	Amazon
# Candidate labels	3	2	3	3
LM learning rate	1×10^{-4}	1×10^{-4}	1×10^{-4}	1×10^{-4}
Retriever learning rate	1×10^{-5}	1×10^{-5}	1×10^{-5}	1×10^{-5}
Weight decay	0	0	0	0

embeddings for each node

$$\texttt{Encoder}_{\psi_1}(t_i) = \mathbf{h}_i^{(0)} \in \mathbb{R}^d, \forall i \in 1, \ldots, n \tag{4.112}$$

In our implementation, the text encoder is all-MiniLM-L6-v2, a member of the Sentence Transformers. Subsequently, we apply a standard graph neural network. In this paper, GraphSAGE [256] is used whose iterative architecture is

$$\mathbf{h}_i^{(l)} = \sigma^{(l)}\left(\texttt{MEAN}\left(\{\mathbf{h}_i^{(l-1)}\} \cup \{\mathbf{h}_j^{(l-1)} : (v_i, v_j) \in \mathcal{E}\}\right) \cdot \mathbf{W}^{(l)}\right) \tag{4.113}$$

where $\sigma^{(l)}$ is the activation function and $\mathbf{W}^{(l)} \in \mathbb{R}^{d \times d}$ is the learnable parameter of each layer. For an L-layer network, in the last layer, $\sigma^{(L)}$ is `softmax` and $\mathbf{W}^{(L)} \in \mathbb{R}^{d \times c}$ so that $\mathbf{h}_i^{(L)} \in \mathbb{R}^c$ is the prediction vector. The typical loss used for training the GNN is negative log-likelihood $\mathcal{L}_{\text{NLL}}(\mathbf{h}_i^{(L)}, y_i)$ for all the nodes in the training set $\mathcal{Y}_{\text{train}}$. The complete set of trainable parameters is denoted as $\psi = \{\psi_1\} \cup \{\mathbf{W}^{(l)}\}_{l=1}^{L}$.

4.3.6 Interpretation of the Distribution Matching Loss

We recap the objective function. For notation brevity, we use t_i to denote the input target node with its pruned candidates: $(t_{\text{target}}, t_{\text{candidates}})$:

$$\text{KL}(\tilde{p}_{\text{LM}}(\cdot|t_i, y_i) \| p_\phi(\cdot|t_i)) \tag{4.114}$$

where the stop gradient operator is removed if we only compute the gradient concerning ϕ and

$$p_\phi(t_j|t_i) = \frac{\exp(s_\phi(t_i, t_j))}{\sum_{t_k \in \mathcal{D}} \exp(s_\phi(t_i, t_k))} \tag{4.115}$$

and

$$\tilde{p}_{\text{LM}}(t_j|t_i, y_i) = \frac{\exp(p_{\text{LM}}(y_i|t_i, t_j))}{\sum_{k \in \mathcal{N}_i} \exp(p_{\text{LM}}(y_i|t_i, t_k))} \tag{4.116}$$

For notation brevity, we replace $\sum_{t_k \in \mathcal{D}}$ with $\sum_z$ if there is no ambiguity. Then

$$\min_\phi \ \text{KL}\big(\tilde{p}_{\texttt{LM}}(\cdot|t_i, y_i) \| p_\phi(\cdot|t_i)\big) \Leftrightarrow \min_\phi - \sum_z \tilde{p}_{\texttt{LM}}(z|t_i, y_i) \log[p_\phi(z|t_i)] \tag{4.117}$$

$$= -\sum_z \tilde{p}_{\texttt{LM}}(z|t_i, y_i) \log \left[\frac{e^{s_\phi(z,t_i)}}{\sum_{z'} e^{s_\phi(z',t_i)}} \right] \tag{4.118}$$

$$= \sum_z \tilde{p}_{\texttt{LM}}(z|t_i, y_i) \log \left[\sum_{z'} e^{s_\phi(z',t_i)} \right] - \sum_z \tilde{p}_{\texttt{LM}}(z|t_i, y_i) s_\phi(z, t_i) \tag{4.119}$$

$$= \log \left[\sum_z e^{s_\phi(z,t_i)} \right] - \sum_z \tilde{p}_{\texttt{LM}}(z|t_i, y_i) s_\phi(z, t_i) \tag{4.120}$$

Hence,

$$\nabla \text{KL} = \frac{\sum_z e^{s_\phi(z,t_i)} \nabla s_\phi(z, t_i)}{\sum_{z'} e^{s_\phi(z',t_i)}} - \sum_z \tilde{p}_{\texttt{LM}}(z|t_i, y_i) \nabla s_\phi(z, t_i) \tag{4.121}$$

$$= \sum_z \left[p_\phi(z|t_i) - \tilde{p}_{\texttt{LM}}(z|t_i, y_i) \right] \nabla s_\phi(z, t_i) \tag{4.122}$$

$$= \sum_z \left[1 - \frac{\tilde{p}_{\texttt{LM}}(z|t_i, y_i)}{p_\phi(z|t_i)} \right] p_\phi(z|t_i) \nabla s_\phi(z, t_i) \tag{4.123}$$

After changing the notation back from $\sum_z$ to $\sum_{t_k \in \mathcal{D}}$, we have

$$\nabla \text{KL} = \sum_{t_k \in \mathcal{D}} \left[1 - \frac{\tilde{p}_{\texttt{LM}}(t_j|t_i, y_i)}{p_\phi(t_j|t_i)} \right] p_\phi(t_j|t_i) \nabla s_\phi(t_j, t_i) \tag{4.124}$$

whose rationale is that if the LM's feedback greatly prefers the neighbor v_j (and its associated text t_j), larger than its probability to be retrieved by the retriever (i.e., $\frac{\tilde{p}_{\texttt{LM}}(t_j|t_i,y_i)}{p_\phi(t_j|t_i)} > 1$), then the similarity score between t_i and t_j will increase, i.e., improve the probability of t_j to be

Table 4.29: Templates used for all datasets.

Template Name	Prompt Text
Citation (Cora, Pubmed, ogbn-arxiv)	`Please classify the following paper into {pruned label candidates} based on the provided information\nTitle: {target node's title}\nContent: {target node's abstract}\nRelated papers: {retrieved nodes' titles}`
Citation, Title Last (Cora, Pubmed, ogbn-arxiv)	`Please classify the following paper into {pruned label candidates} based on the provided information\nContent: {target node's abstract}\nRelated papers: {retrieved nodes' titles}\nTitle: {target node's title}`
Amazon (ogbn-products)	`Please classify the following Amazon product into {pruned label candidates} based on the provided information\nProduct name: {target node's title}\nDescription: {target node's description}\nRelated products: {retrieved nodes' titles}`
Amazon, Title Last (ogbn-products)	`Please classify the following Amazon product into {pruned label candidates} based on the provided information\nDescription: {target node's description}\nRelated products: {retrieved nodes' titles}\nProduct name: {target node's title}`

retrieved.

4.3.7 Templates

Table 4.29 presents templates used in this paper. We design the "Citation" template for the Cora, Pubmed, and ogbn-arxiv datasets and the "Amazon" template for the ogbn-products dataset.

Drawing inspiration from the findings of [267], who demonstrated the efficacy of positioning the title after the main content for particular datasets, we have also introduced two additional template variations: "Citation, Title Last" and "Amazon, Title Last."

CHAPTER 5: GRAPH DISTILLATION

5.1 KERNEL RIDGE REGRESSION-BASED GRAPH DATASET DISTILLATION

5.1.1 Introduction

Graph datasets are indispensable parts of any graph machine learning and graph mining tasks ranging from fraud detection on financial systems [284], and fake account detection on social networks [48], to drug discovery in bioinformatics [285]. Graph neural networks (GNNs), as a family of powerful graph machine learning tools, are becoming critical modules in many graph machine learning systems due to their flexibility and strong expressiveness.

However, similar to other neural network methods, training GNNs with higher model expressiveness usually requires graph datasets with increasing volume [30, 286] and accordingly more intensive computation resource consumption. This limitation naturally leads to a question: *How can we find small, synthetic yet informative datasets to train GNNs with a competitive performance against GNNs trained on large graph datasets?*

The inquiry into the above question has incubated an emerging area named dataset distillation (DD) [287, 288] or dataset condensation (DC) [289]. The core idea of the existing DD or DC methods is to approach the problem under the umbrella of meta-learning and formulate it as a bilevel optimization problem [290]. Specifically, their lower-level problems have training objectives of fitting the synthetic datasets, while the upper-level optimization aims to find the proper synthetic datasets. A variety of settings have been explored for the upper-level problem. For example, Wang et al. [287] set the upper-level problem as the validation loss over the given large dataset; Zhao et al. [289] design a gradient matching loss as the upper-level problem between the gradients on the original dataset and on the synthetic dataset.

The vast majority of the existing works on DD or DC are for tabular data and image data. DD/DC on graph datasets has not been well-studied due to the complex graph structure. To the best of our knowledge, the only graph-level (i.e., a graph is viewed as a data point) dataset distillation work is DosCond [45], which adopts the aforementioned gradient matching strategy [289]. To overcome the huge computation cost for solving the bilevel optimization problem, DosCond [45] proposes a fast but aggressive approximation of the gradient matching loss [289], which only matches the gradients of graph classifiers at initialization. Therefore, there exists unexplored space for performance improvement from the inexact solution of the distillation objective, which is one of our main foci.

Different from previous works, this paper aims to obtain the *exact* solution of the bilevel dataset distillation objective while maintaining computational tractability. To avoid the heavy computation of the bi-level optimization problem, we select kernel ridge regression (KRR) as the classifier whose prediction has a closed form. For implementing KRR on graph-level tasks, a graph kernel is required. Our work selects the recent graph neural tangent kernel (GNTK) [291] for the KRR graph classifier because GNTK describes the training dynamics of GNNs [291]. We name the proposed method Kernel ridge regression-based graph Dataset Distillation (KIDD).

However, due to the inevitable computational intensity of GNTK, naively connecting it with KRR results in low efficiency. To obtain a practically efficient dataset distillation method, we propose a series of novel enhanced designs. A simplified version of GNTK, LiteGNTK, is developed by removing non-linear activations at specific layers. This LiteGNTK can avoid heavy matrix multiplications during the iterative update of the synthetic graphs. By exploiting the close relationship between LiteGNTK and random walk graph kernel [292, 293], we further propose a fast low-rank KIDD variant (KIDD-LR) that boosts the efficiency further. To handle cases where discrete graph topology is required, another variant named KIDD-D is proposed by applying the Gumbel-Max reparameterization trick [294, 295] into our fully differentiable model KIDD.

In comprehensive experiments, our KRR-based model KIDD shows very strong empirical performance over 7 real-world datasets compared with the state-of-the-art methods. In addition, the efficiency study shows our proposed KIDD enjoys comparable efficiency to DosCond, an approximate solver of the distillation objective.

5.1.2 Problem Definition

This section introduces the main notations used throughout this paper. After that, a brief introduction to graph neural networks (GNNs) and graph neural tangent kernel (GNTK) is presented. Then, a formal problem definition is provided.

Notations. We use bold letters for matrices (e.g., $\mathbf{A}$) and column vectors (e.g., $\mathbf{u}$). We use [] as the indices of matrices/vectors. E.g., $\mathbf{A}[i, j]$ represents the entry of matrix $\mathbf{A}$ at the i-th row and j-th column. Superscript $\top$ denotes the transpose of matrices/vectors. We use appropriate subscripts to denote the properties of nodes, graphs, and a set of graphs. For example, $\mathbf{h}_u$ is the representation of node u, $\mathcal{V}_\mathcal{G}$ is the node set of the graph $\mathcal{G}$, and $\mathbf{y}_\mathcal{T}$ is the labels of a graph set $\mathcal{T}$. All the synthetic/distilled graphs and their components are accented with tildes. For example, we notate a synthetic graph as $\tilde{\mathcal{G}} = \{\tilde{\mathbf{A}}, \tilde{\mathbf{X}}\}$ where $\tilde{\mathbf{A}}$ and

$\tilde{\mathbf{X}}$ are its adjacency matrix and node feature matrix, respectively.

Graph Neural Network. The primary operations of GNNs are as follows.
A - Aggregate. To update the node representation of node u, the representations of node u's neighbors are aggregated by the `aggregate` function. A typical `aggregate` function, the summation (with a re-scaling factor of the node u, c_u), is $\mathbf{h}_u \leftarrow c_u \sum_{v \in \mathcal{N}_u \cup \{u\}} \mathbf{h}_v$, where the common choice of $\mathcal{N}_u$ is u's 1-hop neighbors.
B - Update. The representation of a node u can be updated by the `update` function. A simple example is a fully-connected layer with element-wise non-linearity σ (e.g., ReLU [296]) as $\mathbf{h}_u \leftarrow \frac{1}{\sqrt{m}}\sigma(\mathbf{W}\mathbf{h}_u)$, where the m is the output dimension of the $\mathbf{W}\mathbf{h}_u$.
C - Readout. For the graph-level tasks, the graph representation is aggregated over all the nodes in a graph $\mathcal{G}$ by the `readout` function. A typical `readout` function, the summation, is $\mathbf{h}_{\mathcal{G}} = \sum_{v \in \mathcal{V}_{\mathcal{G}}} \mathbf{h}_v$.

GNN variants on graph-level tasks are usually composed of multiple `aggregate` and `update` operations and end with an `readout` operation.

Graph Neural Tangent Kernel. Graph neural tangent kernel (GNTK) [291, 297] is a graph kernel that describes infinitely wide multi-layer GNNs trained by gradient descent through the squared loss. Concretely, the tangent kernel of a GNN f is presented as

$$K_{\boldsymbol{\theta}}(\mathcal{G}, \mathcal{G}') = \left\langle \frac{\partial f(\boldsymbol{\theta}, \mathcal{G})}{\partial \boldsymbol{\theta}}, \frac{\partial f(\boldsymbol{\theta}, \mathcal{G}')}{\partial \boldsymbol{\theta}} \right\rangle, \tag{5.1}$$

where $\boldsymbol{\theta}$ denotes the set of trainable parameters of the GNN. If the width of the GNN is infinite (i.e., $m \to \infty$) and every trainable parameter is an i.i.d. Gaussian random variable, the expectation of the above tangent kernel can be explicitly computed as $K_{\texttt{GNTK}}(\mathcal{G}, \mathcal{G}')$ [291] and it is named GNTK. Du. et al [291] provide a recipe to translate the computation of GNN on node representations from a graph $\mathcal{G}$ into the computation of GNTK on a node covariance matrix from a pair of graphs $\mathcal{G}$ and $\mathcal{G}'$.

Given the graph pair $\mathcal{G}$ and $\mathcal{G}'$ with their node sets $\mathcal{V}_{\mathcal{G}}$ and $\mathcal{V}_{\mathcal{G}'}$, for a pair of nodes $u \in \mathcal{V}_{\mathcal{G}}$ and $u' \in \mathcal{V}_{\mathcal{G}'}$, if we slightly abuse the u and u' as the indices of the matrix, the initial covariance matrix $\boldsymbol{\Sigma}_{\mathcal{G},\mathcal{G}'}$ and the initial GNTK matrix $\boldsymbol{\Theta}_{\mathcal{G},\mathcal{G}'}$ can be computed as

$$\boldsymbol{\Sigma}_{\mathcal{G},\mathcal{G}'}[u, u'] = \boldsymbol{\Theta}_{\mathcal{G},\mathcal{G}'}[u, u'] = \mathbf{x}_u^\top \mathbf{x}_{u'}, \tag{5.2}$$

where $\mathbf{x}$ is the raw node feature. Then, the `aggregate` operation is translated by the GNTK

recipe as

$$\boldsymbol{\Sigma}_{\mathcal{G},\mathcal{G}'}[u,u'] \leftarrow c_u c_{u'} \sum_{v\in\mathcal{N}_u\cup\{u\}} \sum_{v'\in\mathcal{N}_{u'}\cup\{u'\}} \boldsymbol{\Sigma}_{\mathcal{G},\mathcal{G}'}[v,v'] \tag{5.3a}$$

$$\boldsymbol{\Theta}_{\mathcal{G},\mathcal{G}'}[u,u'] \leftarrow c_u c_{u'} \sum_{v\in\mathcal{N}_u\cup\{u\}} \sum_{v'\in\mathcal{N}_{u'}\cup\{u'\}} \boldsymbol{\Theta}_{\mathcal{G},\mathcal{G}'}[v,v'] \tag{5.3b}$$

The `update` operation is translated in the following steps

$$\boldsymbol{\Lambda}_{\mathcal{G},\mathcal{G}'}[u,u'] = \begin{pmatrix} \boldsymbol{\Sigma}_{\mathcal{G},\mathcal{G}}[u,u] & \boldsymbol{\Sigma}_{\mathcal{G},\mathcal{G}'}[u,u'] \\ \boldsymbol{\Sigma}_{\mathcal{G}',\mathcal{G}}[u',u] & \boldsymbol{\Sigma}_{\mathcal{G}',\mathcal{G}'}[u',u'] \end{pmatrix}, \tag{5.4a}$$

$$\dot{\boldsymbol{\Sigma}}_{\mathcal{G},\mathcal{G}'}[u,u'] = \mathbb{E}_{(a,b)\sim\mathcal{N}(0,\boldsymbol{\Lambda}_{\mathcal{G},\mathcal{G}'}[u,u'])} \dot{\sigma}(a)\dot{\sigma}(b), \tag{5.4b}$$

$$\boldsymbol{\Sigma}_{\mathcal{G},\mathcal{G}'}[u,u'] \leftarrow \mathbb{E}_{(a,b)\sim\mathcal{N}(\mathbf{0},\boldsymbol{\Lambda}_{\mathcal{G},\mathcal{G}'}[u,u'])} \sigma(a)\sigma(b), \tag{5.4c}$$

$$\boldsymbol{\Theta}_{\mathcal{G},\mathcal{G}'}[u,u'] \leftarrow \boldsymbol{\Theta}_{\mathcal{G},\mathcal{G}'}[u,u']\dot{\boldsymbol{\Sigma}}_{\mathcal{G},\mathcal{G}'}[u,u'] + \boldsymbol{\Sigma}_{\mathcal{G},\mathcal{G}'}[u,u'], \tag{5.4d}$$

where $\dot{\sigma}(\cdot)$ is the derivative of ReLU. Finally, the `readout` operation is translated as follows, which computes the final GNTK value between the graph pair $\mathcal{G}$ and $\mathcal{G}'$

$$K_{\texttt{GNTK}}(\mathcal{G},\mathcal{G}') = \sum_{u\in\mathcal{V}_{\mathcal{G}},u'\in\mathcal{V}_{\mathcal{G}'}} \boldsymbol{\Theta}_{\mathcal{G},\mathcal{G}'}[u,u']. \tag{5.5}$$

To compute the Eq. (5.4b) and Eq. (5.4c) exactly, according to [291, 298], the following properties of the ReLU activation function should be used. If $\tilde{\boldsymbol{\Lambda}} = \begin{pmatrix} 1 & \lambda \\ \lambda & 1 \end{pmatrix}$,

$$\mathbb{E}_{(a,b)\sim\mathcal{N}(\mathbf{0},\tilde{\boldsymbol{\Lambda}})} \sigma(a)\sigma(b) = \frac{\lambda(\pi - \arccos(\lambda)) + \sqrt{1-\lambda^2}}{2\pi}, \tag{5.6a}$$

$$\mathbb{E}_{(a,b)\sim\mathcal{N}(\mathbf{0},\tilde{\boldsymbol{\Lambda}})} \dot{\sigma}(a)\dot{\sigma}(b) = \frac{\pi - \arccos(\lambda)}{2\pi}. \tag{5.6b}$$

Also, using the homogeneity of ReLU (i.e., for $\forall a \geq 0$, $\sigma(ax) = a\sigma(x)$, and $\dot{\sigma}(ax) = \dot{\sigma}(x)$ we can decompose the $\boldsymbol{\Lambda}_{\mathcal{G},\mathcal{G}'}[u,u'] = \mathbf{D}\tilde{\boldsymbol{\Lambda}}\mathbf{D}$, where $\mathbf{D} = \begin{pmatrix} c_1 & 0 \\ 0 & c_2 \end{pmatrix}$ and get

$$\mathbb{E}_{(a,b)\sim\mathcal{N}(\mathbf{0},\mathbf{D}\tilde{\boldsymbol{\Lambda}}\mathbf{D})} \sigma(a)\sigma(b) = c_1 c_2 \frac{\lambda(\pi - \arccos(\lambda)) + \sqrt{1-\lambda^2}}{2\pi}, \tag{5.7a}$$

$$\mathbb{E}_{(a,b)\sim\mathcal{N}(\mathbf{0},\mathbf{D}\tilde{\boldsymbol{\Lambda}}\mathbf{D})} \dot{\sigma}(a)\dot{\sigma}(b) = \frac{\pi - \arccos(\lambda)}{2\pi}. \tag{5.7b}$$

Graph Dataset Distillation. In this paper, we study the dataset distillation problem for graph classification tasks. Specifically, we aim to synthesize a small number of informative graphs to empower the training of graph classifiers. The problem is formally defined as follows.

Problem 5.1. Graph dataset distillation
Given: a set of target training graphs $\mathcal{T} = \{(\mathcal{G}_i, y_i)\}_0^{n_\mathcal{T}-1}$.
Find: a set of synthetic training graphs $\mathcal{S} = \{(\tilde{\mathcal{G}}_i, \tilde{y}_i)\}_0^{n_\mathcal{S}-1}$ such that the GNN trained over $\mathcal{S}$ can obtain competitive performance with GNN trained over $\mathcal{T}$.

The overall procedure is two-step. First, a distillation method (including a **distillation classifier**, introduced in Section 5.1.3) is applied to synthesize $\mathcal{S}$ from $\mathcal{T}$. Then, a **downstream classifier** is trained over $\mathcal{S}$ and reports its performance on a test set $\mathcal{U}$ which has no overlap with $\mathcal{S}$ or $\mathcal{T}$. The downstream classifier's test performance is the metric of the distilled training set $\mathcal{S}$, and in this paper, the downstream classifier is selected from graph-level GNNs (e.g., GIN [299]) and will be introduced in detail in Section 5.1.4.

5.1.3 Proposed Method

This section introduces the objective formulation, the proposed model (KIDD), and the corresponding enhancements.

Optimization Objective. An ideal distilled graph dataset $\mathcal{S}$ for the target graph dataset $\mathcal{T}$ should minimize the following optimization objective.

$$\min_{\mathcal{S}} \quad \left|\mathbb{E}_\mathcal{U}\mathcal{L}(\mathcal{U}, f(\boldsymbol{\theta}^*_\mathcal{S})) - \mathbb{E}_\mathcal{U}\mathcal{L}(\mathcal{U}, f(\boldsymbol{\theta}^*_\mathcal{T}))\right|, \tag{5.8a}$$

$$\text{s.t.} \quad \boldsymbol{\theta}^*_\mathcal{S} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{S}, f(\boldsymbol{\theta})), \tag{5.8b}$$

$$\boldsymbol{\theta}^*_\mathcal{T} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{T}, f(\boldsymbol{\theta})), \tag{5.8c}$$

where $\mathcal{U}$ is the test set sampled from the true graph distribution. The above formula suggests the graph classifiers trained on the target dataset (i.e., $f(\boldsymbol{\theta}^*_\mathcal{T})$ from Eq.(5.8c)) and on the synthetic dataset (i.e., $f(\boldsymbol{\theta}^*_\mathcal{S})$ from Eq.(5.8b)) should have similar expected test loss (i.e., Eq. (5.8a)). Notice here the graph classifier f is for the distillation purpose, i.e., synthesizing the dataset $\mathcal{S}$. Thus, we name f as the **distillation classifier** in this paper. Naturally, Eq. (5.8a) implies *the best choice of the distillation classifier f should be the same as the downstream classifier.* That is because, if the downstream classifier is g, the test error

$\mathbb{E}_{\mathcal{U}}\mathcal{L}(\mathcal{U}, g(\boldsymbol{\theta}^*_{\mathcal{S}}))$ is minimized when $f = g$, considering (1) $|\mathcal{T}| \gg |\mathcal{S}|$ and (2) empirically, $\mathbb{E}_{\mathcal{U}}\mathcal{L}(\mathcal{U}, f(\boldsymbol{\theta}^*_{\mathcal{S}})) \geq \mathbb{E}_{\mathcal{U}}\mathcal{L}(\mathcal{U}, f(\boldsymbol{\theta}^*_{\mathcal{T}}))$.

However, as the true graph distribution is not accessible, a feasible optimization goal is to replace the unknown test set $\mathcal{U}$ with the large target training set $\mathcal{T}$ as follows.

$$\min_{\mathcal{S}} \quad |\mathcal{L}(\mathcal{T}, f(\boldsymbol{\theta}^*_{\mathcal{S}})) - \mathcal{L}(\mathcal{T}, f(\boldsymbol{\theta}^*_{\mathcal{T}}))|, \tag{5.9}$$

where $\boldsymbol{\theta}^*_{\mathcal{T}}$ and $\boldsymbol{\theta}^*_{\mathcal{S}}$ are from Eq. (5.8c) and Eq. (5.8b). As the $\boldsymbol{\theta}^*_{\mathcal{T}}$ is the minimizer over the $\mathcal{L}(\mathcal{T}, f(\boldsymbol{\theta}))$, to minimize Eq. (5.9), our objective is equivalent to minimizing $\mathcal{L}(\mathcal{T}, f(\boldsymbol{\theta}^*_{\mathcal{S}}))$. The objective function can be re-written as follows,

$$\min_{\mathcal{S}} \quad \mathcal{L}(\mathcal{T}, f(\boldsymbol{\theta}^*_{\mathcal{S}})), \tag{5.10a}$$

$$\text{s.t.} \quad \boldsymbol{\theta}^*_{\mathcal{S}} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{S}, f(\boldsymbol{\theta})), \tag{5.10b}$$

which can be interpreted as finding a synthetic training set $\mathcal{S}$ such that the trained model $f(\boldsymbol{\theta}^*_{\mathcal{S}})$ over $\mathcal{S}$ has a small loss over the validation set $\mathcal{T}$. The above objective is a bilevel optimization problem [290] whose exact solutions [116, 150, 151, 300, 301, 302] is computationally expensive or even intractable, especially when the objective functions are not convex. Thus, even though the best distillation classifier f is the downstream GNN classifier itself, its non-convex optimization objective makes the distillation problem costly to be solved. In this paper, we resort to a kernel-based method that renders a tractable and exact solution to the above objective function.

Graph Kernel Ridge Regression. To avoid the expensive hyper-gradient computation of the bilevel optimization problem, a strategy is that if the lower-level problem has a closed-form solution, plugging the lower-level solution into the upper-level objective can largely simplify the optimization objective [108, 303] into a single-level problem. Kernel ridge regression (KRR) can yield such a closed-form solution. If f is instantiated as the KRR and the squared loss is applied, the Eq. (5.10a) and Eq. (5.10b) can be instantiated as

$$\min_{\mathcal{S}} \quad \mathcal{L}_{\texttt{KRR}} = \frac{1}{2}\left\|\mathbf{y}_{\mathcal{T}} - \mathbf{K}_{\mathcal{T}\mathcal{S}}(\mathbf{K}_{\mathcal{S}\mathcal{S}} + \epsilon\mathbf{I})^{-1}\mathbf{y}_{\mathcal{S}}\right\|^2, \tag{5.11}$$

where $\epsilon > 0$ is a KRR hyper-parameter, $\mathbf{K}_{\mathcal{T}\mathcal{S}}$ and $\mathbf{K}_{\mathcal{S}\mathcal{S}}$ are the kernel matrices. E.g., $\mathbf{K}_{\mathcal{T}\mathcal{S}}[i,j] = K(\mathcal{G}_i, \tilde{\mathcal{G}}_j)$, $\mathbf{K}_{\mathcal{S}\mathcal{S}}[i,j] = K(\tilde{\mathcal{G}}_i, \tilde{\mathcal{G}}_j)$, $\mathcal{G}_i \in \mathcal{T}$, $\tilde{\mathcal{G}}_i$, $\tilde{\mathcal{G}}_j \in \mathcal{S}$ with K as a specific kernel function (e.g., random walk graph kernel [292, 293]). $\mathbf{y}_{\mathcal{T}}$ and $\mathbf{y}_{\mathcal{S}}$ are the concatenated graph labels from $\mathcal{T}$ and $\mathcal{S}$.

However, there are two concerns about the above optimization objective. First (*model gap*), in terms of expressive power and empirical performance, GNNs outperform most, if not all, classic graph kernel methods on graph classification tasks [299]. It leads to a concern that *whether the graph dataset $\mathcal{S}$ distilled through the graph kernel-based kernel ridge regression can empower the training of downstream GNNs.* Second (*informative distillation*), as the size of the distilled graph dataset is small, how to ensure the distilled graphs are sufficiently informative to capture the critical information from the original training set.

As a response to the first concern and to bridge the gap between the distillation classifier and the downstream GNN classifier, a recent graph kernel is adopted which is named graph neural tangent kernel (GNTK) [291]. GNTK (1) measures the graph pair similarity by mapping a graph into an infinite-dimensional GNN gradient vector and (2) describes the training dynamics of the corresponding GNN. Thus, GNTK-based kernel ridge regression is promising to distill generalizable graph datasets for downstream GNN classifiers. The specific computation of GNTK is introduced in Section 5.1.2.

As a response to the second concern and to ensure informative distilled graphs, we propose a distillation regularization term as

$$\mathcal{L}_{\texttt{reg}} = \left\| \hat{\mathbf{K}}_{\mathcal{SS}} - \mathbf{I} \right\|_F^2. \tag{5.12}$$

The final objective $\mathcal{L}$ is the weighted sum of $\mathcal{L}_{\texttt{KRR}}$ and $\mathcal{L}_{\texttt{reg}}$ as

$$\min_{\mathcal{S}} \quad \mathcal{L} = \mathcal{L}_{\texttt{KRR}} + \gamma \mathcal{L}_{\texttt{reg}}. \tag{5.13}$$

$\hat{\mathbf{K}}_{\mathcal{SS}}$ is the normalized kernel matrix between synthetic graphs whose entry $\hat{\mathbf{K}}_{\mathcal{SS}}[i,j] = \frac{\mathbf{K}_{\mathcal{SS}}[i,j]}{\sqrt{\mathbf{K}_{\mathcal{SS}}[i,i]}\sqrt{\mathbf{K}_{\mathcal{SS}}[j,j]}}$. γ is a hyper-parameter to trade-off the KRR classification loss $\mathcal{L}_{\texttt{KRR}}$ and the regularization loss $\mathcal{L}_{\texttt{reg}}$. The key idea of $\mathcal{L}_{\texttt{reg}}$ is to force the normalized kernel matrix $\hat{\mathbf{K}}_{\mathcal{SS}}$ to be an identity matrix so that the synthetic graphs are more orthogonal to each other in the kernel space.

Given the optimization objective as Eq. (5.13) shows, the gradient of the objective with respect to the graphs from $\mathcal{S}$ are computed, i.e., $\frac{\partial \mathcal{L}}{\partial \tilde{\mathcal{G}}_i}$, $\forall \tilde{\mathcal{G}}_i \in \mathcal{S}$. We provide some details here. First, in our implementation, we compute the gradient with respect to the graph adjacency matrix $\tilde{\mathbf{A}}$ and node feature matrix $\tilde{\mathbf{X}}$ for all the graphs from $\mathcal{S} = \{(\tilde{\mathcal{G}}_i, \tilde{y}_i)\}_0^{n_{\mathcal{S}}-1}$. Second, any gradient-based optimizer can be applied, e.g., Adam [304]. Third, the synthetic set $\mathcal{S}$ can be initialized by sampling the target set $\mathcal{T}$ or fully initialized randomly. In this work, we initialize the synthetic set $\mathcal{S}$ by sampling $\mathcal{T}$. We name our proposed graph dataset distillation method KIDD. Next, we present the enhancements designed for KIDD to

improve its efficiency greatly and handle discrete graph structures.

Model Enhancements. The existing computation of GNTK [291] is still expensive, especially for our gradient descent-based dataset distillation scenario. That is because, in every iteration, the kernel matrices $\mathbf{K}_{\mathcal{SS}}$ and $\mathbf{K}_{\mathcal{TS}}$ need to be re-computed. Additionally, it is non-trivial to synthesize discrete graph topology by the gradient descent-based method, which is required in certain cases. This section introduces several enhancements that systematically improve KIDD's computational feasibility and functionality.

LiteGNTK. As we introduced in Section 5.1.2, Du. et al [291] provides a recipe to translate a specific instantiation of GNN into its corresponding GNTK. The GNTK used in the existing literature [291] is based on the typical GNNs with a non-linear activation function at every layer (e.g., GCN [20] and GIN [299]). Hence, every layer of their corresponding GNTK instantiations contains a translated `update` operation (i.e., Eq. (5.4a)-(5.4d)). Recently, extensive empirical studies on GNNs show that non-linearity is not a necessity for every GNN layer, and removing the non-linearity from the last several layers can speed up the computation [22] without sacrificing or even improving the model performance [305]. Thus, we propose to remove the `update` operation from all the GNTK layers except the first one and use the following LiteGNTK for our kernel ridge regression-based graph dataset distillation. Given two graphs $\mathcal{G} = \{\mathbf{A}, \mathbf{X}\}$ and $\tilde{\mathcal{G}} = \{\tilde{\mathbf{A}}, \tilde{\mathbf{X}}\}$, the LiteGNTK $K_{\texttt{LiteGNTK}}$ between this graph pair is computed as

$$\mathbf{\Sigma}_{\mathcal{G},\tilde{\mathcal{G}}} = \mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}} = \mathbf{X}\tilde{\mathbf{X}}^\top, \tag{5.14a}$$

$$\mathbf{\Sigma}_{\mathcal{G},\tilde{\mathcal{G}}}, \mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}} \leftarrow \texttt{GNTK-update}\left(\mathbf{\Sigma}_{\mathcal{G},\tilde{\mathcal{G}}}, \mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}}\right), \tag{5.14b}$$

$$\mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}} \leftarrow (\mathbf{c}_{\mathcal{G}}\mathbf{c}_{\tilde{\mathcal{G}}}^\top) \odot \left(\mathbf{A}^k \mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}} (\tilde{\mathbf{A}}^\top)^k\right), \tag{5.14c}$$

$$K_{\texttt{LiteGNTK}}(\mathcal{G}, \tilde{\mathcal{G}}) = \sum_{u,\tilde{u}} \mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}}[u, \tilde{u}], \tag{5.14d}$$

where $\odot$ is the Hadamard product; `GNTK-update` is an alias of Eq. (5.4a)-(5.4d); $\mathbf{c}_{\mathcal{G}} = \texttt{vec}(\{c_u^k | u \in \mathcal{G}\})$ is the scaling vector whose elements are the k-powered re-scaling factor of every node from $\mathcal{G}$; $\mathbf{c}_{\tilde{\mathcal{G}}}$ is constructed similarly by the nodes from $\tilde{\mathcal{G}}$. Finally, the kernel value between the graph $\mathcal{G}$ and $\tilde{\mathcal{G}}$ are computed by the Eq. (5.14d) which is the same as the GNTK `readout` function. LiteGNTK is used for all our proposed distillation models.

Advantages of LiteGNTK. First, the original GNTK [291] contains non-linearity in each of the k layers, which requires the computation of $\mathbf{A}\mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}}\tilde{\mathbf{A}}^\top$ for k times during every update of the synthetic graphs. In comparison, for LiteGNTK, $\mathbf{A}^k$ and $\tilde{\mathbf{A}}^k$ can be precomputed,

especially for the $\mathbf{A}$s from the target dataset $\mathcal{T}$ which are not updated throughout the whole distillation procedure. Second, the LiteGNTK is closely related to the random walk graph kernel [292, 293], which is shown in our following proposition. As both LiteGNTK and random walk graph kernel include power iterations of adjacency matrices, this reveals the possibility to develop a faster LiteGNTK in light of the speedup of the random walk graph kernel.

Proposition 5.1. LiteGNTK is a generalized instantiation of the random walk graph kernel [292, 293].

Proof. Random walk graph kernel (RWK) counts the number of length-k common paths between graph $\mathcal{G}$ and $\tilde{\mathcal{G}}$ as

$$K_{\texttt{RWK}}\left(\mathcal{G},\tilde{\mathcal{G}}\right)=(\mathbf{q}\otimes\tilde{\mathbf{q}})^{\top}\left(\mathbf{A}\otimes\tilde{\mathbf{A}}\right)^{k}(\mathbf{p}\otimes\tilde{\mathbf{p}}), \tag{5.15}$$

where $\mathbf{p}$ ($\tilde{\mathbf{p}}$) and $\mathbf{q}$ ($\tilde{\mathbf{q}}$) are the starting and stopping probability vectors of graph $\mathcal{G}$ ($\tilde{\mathcal{G}}$). Thus, Eq. (5.15) can be written as

$$\begin{aligned}
K_{\texttt{RWK}} &\underset{(1)}{=} (\mathbf{q}\otimes\tilde{\mathbf{q}})^{\top}(\mathbf{A}\otimes\tilde{\mathbf{A}})^{(k-1)}\texttt{vec}\Big(\mathbf{A}\Big(\texttt{vec}^{-1}(\mathbf{p}\otimes\tilde{\mathbf{p}})\Big)\tilde{\mathbf{A}}^{\top}\Big), && \text{(5.16a)}\\
&\underset{(2)}{=} (\mathbf{q}\otimes\tilde{\mathbf{q}})^{\top}\texttt{vec}\Big(\mathbf{A}^{k}(\mathbf{p}\tilde{\mathbf{p}}^{\top})(\tilde{\mathbf{A}}^{\top})^{k}\Big), && \text{(5.16b)}\\
&\underset{(3)}{=} \mathbf{q}^{\top}\mathbf{A}^{k}(\mathbf{p}\tilde{\mathbf{p}}^{\top})(\tilde{\mathbf{A}}^{\top})^{k}\tilde{\mathbf{q}}, && \text{(5.16c)}\\
&\underset{(4)}{=} \sum_{u,\tilde{u}}\Big((\mathbf{q}\tilde{\mathbf{q}}^{\top})\odot\Big(\mathbf{A}^{k}(\mathbf{p}\tilde{\mathbf{p}}^{\top})(\tilde{\mathbf{A}}^{\top})^{k}\Big)\Big)[u,\tilde{u}], && \text{(5.16d)}
\end{aligned}$$

where $\otimes$ is the Kronecker product, $\odot$ is the Hadamard product, $\texttt{vec}$ is the vectorization operator, and $\texttt{vec}^{-1}$ is the inverse operator of $\texttt{vec}$ (i.e., reshaping a vector into a matrix). Note here the $\texttt{vec}$ and $\texttt{vec}^{-1}$ follow the row-first order as many scientific computing packages do (e.g., PyTorch's reshape function). The above step (1), (2), and (3) are based on the property of the Kronecker product: $(\mathbf{A}\otimes\mathbf{B})\mathbf{v}=\texttt{vec}\Big(\mathbf{A}\Big(\texttt{vec}^{-1}(\mathbf{v})\Big)\mathbf{B}^{\top}\Big)$. Step (4) is because $\mathbf{v}^{\top}\mathbf{A}\mathbf{v}=\sum_{i,j}\mathbf{v}[i]\mathbf{A}[i,j]\mathbf{v}[j]=\sum_{i,j}\mathbf{v}[i]\mathbf{v}[j]\mathbf{A}[i,j]=\sum_{i,j}\Big((\mathbf{v}\mathbf{v}^{\top})\odot\mathbf{A}\Big)[i,j]$. Here, the matrix $\mathbf{p}\tilde{\mathbf{p}}^{\top}$ describes the node pair co-starting probability from graphs $\mathcal{G}$ and $\tilde{\mathcal{G}}$ which can be generalized as the GNTK-updated node pair co-variance matrix $\mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}}$ from Eq. (5.14b). The matrix $\mathbf{q}\tilde{\mathbf{q}}^{\top}$ is a node pair co-stopping probability matrix which can be generalized as the node pair weighting matrix $\mathbf{c}_{\mathcal{G}}\mathbf{c}_{\tilde{\mathcal{G}}}^{\top}$ from Eq. (5.14c). Thus, we conclude that the LiteGNTK $K_{\texttt{LiteGNTK}}$ is the generalization of the random walk graph kernel $K_{\texttt{RWK}}$. QED.

Low-Rank Speed-Up. Proposition 5.1 reveals the connection between the random walk graph kernel and the LiteGNTK, which inspires us to apply the fast strategy [293] proposed for the random walk graph kernel into the computation of LiteGNTK. Real-world graphs are known for their intrinsic low-rank topology [100], which can significantly speed up the power iterations of the adjacency matrices using their low-rank representations. To be specific, for learning the topology of synthetic graphs, instead of optimizing the objective Eq. (5.13) with respect to the synthetic graph's adjacency matrix $\tilde{\mathbf{A}}$, we directly optimize its low-rank decomposed matrix $\tilde{\mathbf{U}},\ \tilde{\mathbf{V}} \in \mathbb{R}^{n\times r}$ where r is the rank of the synthetic graphs.

Notice that in this paper, we retain the adjacency matrices of the original training graphs $\mathcal{G} \in \mathcal{T}$ and only learn the low-rank matrices of the synthetic graphs $\tilde{\mathcal{G}} \in \mathcal{S}$. There are 2 reasons: (1) the original training graphs are the distillation target, whose topology information should be kept intact; (2) the original training graphs are usually sparse but the synthetic graphs could be dense due to the gradient descent update, which leads to heavy computation (e.g., matrix multiplication). Since the $\mathbf{K}_{\mathcal{TS}}$ has much more entries than the matrix $\mathbf{K}_{\mathcal{SS}}$, in the following part, we analyze the computation of the kernel value between $\mathcal{G} = \{\mathbf{A}, \mathbf{X}\} \in \mathcal{T}$ and $\tilde{\mathcal{G}} = \{\tilde{\mathbf{U}}, \tilde{\mathbf{V}}, \tilde{\mathbf{X}}\} \in \mathcal{S}$, i.e., an entry of the matrix $\mathbf{K}_{\mathcal{TS}}$. For computing entries of $\mathbf{K}_{\mathcal{SS}}$, it can be analyzed similarly and is omitted for brevity.

Given $\mathcal{G} = \{\mathbf{A}, \mathbf{X}\} \in \mathcal{T}$ and $\tilde{\mathcal{G}} = \{\tilde{\mathbf{U}}, \tilde{\mathbf{V}}, \tilde{\mathbf{X}}\} \in \mathcal{S}$, to compute their corresponding LiteGNTK value, Eq. (5.14c) is modified as follows,

$$\mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}} \leftarrow \left(\mathbf{c}_{\mathcal{G}}\mathbf{c}_{\tilde{\mathcal{G}}}^{\top}\right) \odot \left(\mathbf{A}^k \mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}} \left(\tilde{\mathbf{V}}\tilde{\mathbf{U}}^{\top}\right)^k\right), \tag{5.17}$$

while Eq. (5.14a), (5.14b), (5.14d) stay unchanged. The following lemma shows such a minor change can greatly improve the model's efficiency during both the forward computation and the gradient backward propagation. Notice that the following analysis focuses on the computation efficiency of the key operations containing the synthetic graph topology variables $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$.

Lemma 5.1. (Time Complexity) Assume both $\mathcal{G}$ and $\tilde{\mathcal{G}}$ have n nodes, i.e., $\mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}} \in \mathbb{R}^{n\times n}$, the time complexity of computing $\mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}} \left(\tilde{\mathbf{V}}\tilde{\mathbf{U}}^{\top}\right)^k$ is $O(rn^2)$. The time complexity of computing $\frac{\partial(\tilde{\mathbf{V}}\tilde{\mathbf{U}}^{\top})^k}{\partial\tilde{\mathbf{U}}}$ and $\frac{\partial(\tilde{\mathbf{V}}\tilde{\mathbf{U}}^{\top})^k}{\partial\tilde{\mathbf{V}}}$ is $O(r^3n^3)$.

Proof. $\mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}}(\tilde{\mathbf{V}}\tilde{\mathbf{U}}^{\top})^k$ can be rewritten as $\mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}}\tilde{\mathbf{V}}(\tilde{\mathbf{U}}^{\top}\tilde{\mathbf{V}})^{k-1}\tilde{\mathbf{U}}^{\top}$. Thus, the complexity of computing $(\tilde{\mathbf{U}}^{\top}\tilde{\mathbf{V}})^{k-1}$ is $O(rn^2 + (k-2)r^3)$, the complexity of multiply $\mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}}$, $\tilde{\mathbf{V}}$, $(\tilde{\mathbf{U}}^{\top}\tilde{\mathbf{V}})^{k-1}$, $\tilde{\mathbf{U}}^{\top}$ from the left-hand side is $O(2rn^2 + r^2n)$. Hence, putting everything together the time complexity of computing $\mathbf{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}}(\tilde{\mathbf{V}}\tilde{\mathbf{U}}^{\top})^k$ is $O(3rn^2 + r^2n + (k-2)r^3)$ which can be shortened as $O(rn^2)$ given $r \ll n$ and $k \ll n$.

$\frac{\partial(\tilde{\mathbf{V}}\tilde{\mathbf{U}}^\top)^k}{\partial\tilde{\mathbf{U}}} = \frac{\partial\tilde{\mathbf{V}}(\tilde{\mathbf{U}}^\top\tilde{\mathbf{V}})^{k-1}\hat{\mathbf{U}}^\top}{\partial\tilde{\mathbf{U}}^\top\tilde{\mathbf{V}}}\frac{\partial\tilde{\mathbf{U}}^\top\tilde{\mathbf{V}}}{\partial\tilde{\mathbf{U}}} + \frac{\partial\tilde{\mathbf{V}}(\tilde{\mathbf{U}}^\top\tilde{\mathbf{V}})^{k-1}\hat{\mathbf{U}}^\top}{\partial\hat{\mathbf{U}}}$, where $\hat{\mathbf{U}} = \tilde{\mathbf{U}}$. It is a fact that, for any matrix $\mathbf{M} \in \mathbb{R}^{r\times r}$

$$\begin{aligned}\frac{\partial\mathbf{M}^k[i,j]}{\partial\mathbf{M}[s,t]} &= \frac{\partial\sum_{l_1,\ldots,l_{k-1}}\mathbf{M}[i,l_1]\mathbf{M}[l_1,l_2]\ldots\mathbf{M}[l_{k-1},j]}{\partial\mathbf{M}[s,t]}, \\ &= \left(\sum_{l=0}^{k-1}\mathbf{M}^l\otimes(\mathbf{M}^\top)^{k-1-l}\right)[ri+j, rs+t].\end{aligned} \tag{5.18}$$

Thus, if we represent $\frac{\partial(\tilde{\mathbf{U}}^\top\tilde{\mathbf{V}})^{k-1}}{\partial\tilde{\mathbf{U}}^\top\tilde{\mathbf{V}}}$ in the shape of $\mathbb{R}^{r^2\times r^2}$ it can be computed as $\sum_{l=0}^{k-2}(\tilde{\mathbf{U}}^\top\tilde{\mathbf{V}})^l\otimes(\tilde{\mathbf{V}}^\top\tilde{\mathbf{U}})^{k-2-l}$ whose time complexity is $O((k-1)(2r^2n+r^4))$. The time complexity of computing $\frac{\partial\tilde{\mathbf{U}}^\top\tilde{\mathbf{V}}}{\partial\tilde{\mathbf{U}}}$ is $O(r^3n)$. The complexity of multiply $\frac{\partial\tilde{\mathbf{V}}(\tilde{\mathbf{U}}^\top\tilde{\mathbf{V}})^{k-1}\hat{\mathbf{U}}^\top}{\partial\tilde{\mathbf{U}}^\top\tilde{\mathbf{V}}}$ and $\frac{\partial\tilde{\mathbf{U}}^\top\tilde{\mathbf{V}}}{\partial\tilde{\mathbf{U}}}$ is $O(r^3n^3)$. The time complexity of computing $\frac{\partial\tilde{\mathbf{V}}(\tilde{\mathbf{U}}^\top\tilde{\mathbf{V}})^{k-1}\hat{\mathbf{U}}^\top}{\partial\hat{\mathbf{U}}}$ is $O(rn^3)$. Consequently, put everything together, the time complexity of computing $\frac{\partial(\tilde{\mathbf{V}}\tilde{\mathbf{U}}^\top)^k}{\partial\tilde{\mathbf{U}}}$ is $O(r^3n^3)$ given $r \ll n$ and $k \ll n$.

The time complexity of computing $\frac{\partial\boldsymbol{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}}(\tilde{\mathbf{V}}\tilde{\mathbf{U}}^\top)^k}{\partial\tilde{\mathbf{V}}}$ can be analyzed similarly and is omitted for brevity. QED.

As a comparison, without this low-rank speed-up technique, the time complexity of computing $\boldsymbol{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}}\tilde{\mathbf{A}}^k$ is $O(kn^3)$ and time complexity of computing $\frac{\partial\tilde{\mathbf{A}}^k}{\partial\tilde{\mathbf{A}}}$ is $O(kn^4)$ if $\tilde{\mathbf{A}}$ is a dense matrix after gradient descent-based updating. Thus, considering $r \ll n$, our proposed low-rank variant can significantly speed up the computation. In addition, clearly, the space complexity for storing every synthetic graph's topology drops to $O(2nr)$ if the low-rank technique is applied; otherwise, it is $O(n^2)$.

Benefiting from the close connection between the random walk graph kernel and LiteGNTK, the following lemma gives an error bound on applying the low-rank technique to LiteGNTK. For brevity, the lemma 5.2 only analyzes the case where synthetic graphs are undirected (i.e., $\tilde{\mathbf{A}}$ is symmetric).

Lemma 5.2. (Error Bound) Given a target graph $\mathcal{G} = \{\mathbf{A}, \mathbf{X}\}$, a synthetic graph $\tilde{\mathcal{G}} = \{\tilde{\mathbf{A}}, \tilde{\mathbf{X}}\}$ and $\tilde{\mathcal{G}}$'s rank-r representation $\tilde{\mathcal{G}}_r = \{\tilde{\mathbf{U}}\tilde{\mathbf{U}}^\top, \tilde{\mathbf{X}}\}$, if we assume both $\mathcal{G}$ and $\tilde{\mathcal{G}}$ have n nodes, $\mathcal{G}$ has m edges, and $\mathbf{c}_{\mathcal{G}} = \mathbf{c}_{\tilde{\mathcal{G}}} = \mathbf{c}_{\tilde{\mathcal{G}}_r} = \mathbf{1}$ are all-one vectors, the error of the LiteGNTK value after applying the low-rank speed-up can be bounded by

$$K_{\texttt{LiteGNTK}}(\mathcal{G},\tilde{\mathcal{G}}) - K_{\texttt{LiteGNTK}}(\mathcal{G},\tilde{\mathcal{G}}_r) \leq nm^{\frac{k}{2}}||\boldsymbol{\Theta}||_F\sum_{i=r+1}^{n}\left|\tilde{\lambda_i}^k\right|, \tag{5.19}$$

where $\boldsymbol{\Theta} = \boldsymbol{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}} = \boldsymbol{\Theta}_{\mathcal{G},\tilde{\mathcal{G}}_r}$ is the output of Eq. (5.14b) and $\tilde{\lambda_i}$ is the i-th largest eigenvalue of $\tilde{\mathbf{A}}$.

Proof. The difference between $\mathbf{K}_{\texttt{LiteGNTK}}(\mathcal{G}, \tilde{\mathcal{G}})$ and $\mathbf{K}_{\texttt{LiteGNTK}}(\mathcal{G}, \tilde{\mathcal{G}}_r)$ can be presented as

$$\begin{aligned}
&\left|\mathbf{c}_{\tilde{\mathcal{G}}}^\top \mathbf{A}^k \mathbf{\Theta} (\tilde{\mathbf{A}}^k - (\tilde{\mathbf{U}}\tilde{\mathbf{U}}^\top)^k) \mathbf{c}_{\mathcal{G}}\right| \\
&= \left|\mathbf{c}_{\tilde{\mathcal{G}}}^\top \mathbf{A}^k \mathbf{\Theta} (\sum_{i=r+1}^{n} \tilde{\lambda_i}^k \tilde{\mathbf{u}}_i \tilde{\mathbf{u}}_i^\top) \mathbf{c}_{\mathcal{G}}\right| \\
&\leq \|\mathbf{c}_{\mathcal{G}}\|_2 \|\mathbf{c}_{\tilde{\mathcal{G}}}\|_2 ||\mathbf{A}||_F^k \|\mathbf{\Theta}\|_F \left\|\sum_{i=r+1}^{n} \tilde{\lambda_i}^k \tilde{\mathbf{u}}_i \tilde{\mathbf{u}}_i^\top\right\|_F \\
&\leq n m^{\frac{k}{2}} \|\mathbf{\Theta}\|_F \sum_{i=r+1}^{n} \left|\tilde{\lambda_i}^k\right|
\end{aligned} \tag{5.20}$$

where $\tilde{\mathbf{u}}_i$ is the i-th unit eigenvector of $\tilde{\mathbf{A}}$, and the last inequality holds because (1) $\{\mathbf{u}_i\}$ are the normalized eigenvectors, and (2) $||\sum_i a_i \mathbf{u}_i \mathbf{u}_i^\top||_F = \sqrt{\texttt{trace}(\sum_i a_i^2 \mathbf{u}_i \mathbf{u}_i^\top)}$ $= \sqrt{\sum_i a_i^2 \texttt{trace}(\mathbf{u}_i \mathbf{u}_i^\top)} \leq \sum_i |a_i|$. The above bound can be further simplified by limiting $\tilde{\mathbf{X}}$, $\mathbf{X}$, and $\mathbf{\Theta}$ to special cases but we keep this form for generality. QED.

Our distillation model equipped with this low-rank speed-up technique is named KIDD-LR, and a step-by-step algorithm to distill a graph dataset is presented in Algorithm 5.1. In addition, as Lemma 5.2 shows, if the low-rank assumption of the synthetic graphs holds, the proposed KIDD-LR still provides an exact solution; otherwise, it provides an approximate solution.

Discrete Synthetic Graphs. Despite the great efficacy of our proposed low-rank speed-up method, the synthetic graph topology $\tilde{\mathbf{U}}\tilde{\mathbf{V}}^\top$ is weighted and even needs clipping to be non-negative in the inference phase. For the case where discrete unweighted graphs are needed, i.e., all the entries all either 0 or 1, a discrete variant of our method is proposed. The strategy is to model every pair of nodes as an independent Bernoulli variable [45, 54, 306] (e.g., $b_{u,v} \in [0, 1]$) such that $\tilde{\mathbf{A}}[u, v] \sim \texttt{Bernoulli}(b_{u,v})$. As the sampling process is not differentiable, we utilize the Gumbel-Max reparametrization trick [45, 294, 295], and the adjacency matrix is computed as

$$\tilde{\mathbf{A}}[u, v] = \texttt{sigmoid}((\log \delta - \log(1 - \delta) + b_{u,v})/\tau), \tag{5.21}$$

where $\delta \sim \texttt{Uniform}(0, 1)$, and τ is a temperature hyperparameter. If $\tau \to 0$, $\tilde{\mathbf{A}}[u, v]$ will be binary. Then, instead of modeling graph topology by its adjacency matrix, we can describe it by a corresponding Bernoulli parameter matrix $\tilde{\mathbf{B}}$ whose entries are edge-existing Bernoulli

variables, i.e., $\tilde{\mathbf{B}}[u,v] = b_{u,v}$. As the gradient $\frac{\partial \tilde{\mathbf{A}}[u,v]}{\partial \tilde{\mathbf{B}}[u,v]}$ is well-defined, we can optimize $\tilde{\mathbf{B}}$ in an end-to-end fashion by the gradient descent. During the inference phase, we can set $\tilde{\mathbf{A}}[u,v] = 1$ if $\texttt{sigmoid}(b_{u,v}) > 0.5$; otherwise, $\tilde{\mathbf{A}}[u,v] = 0$.

Our distillation model equipped with this Gumbel-Max technique is named KIDD-D. Notice that even though we can apply a similar low-rank decomposition trick to decompose the Bernoulli parameter matrix $\tilde{\mathbf{B}} = \tilde{\mathbf{U}}\tilde{\mathbf{V}}^\top$, it will not improve the model's efficacy as we introduced in the above subsection. That is because the efficiency improvement is mainly from the re-ordering of the computation of the power iteration (i.e., $\left(\tilde{\mathbf{U}}\tilde{\mathbf{V}}^\top\right)^k$) which cannot be applied to the power iteration of the sampled adjacency matrix (i.e., $\left(\texttt{sample}\left(\tilde{\mathbf{U}}\tilde{\mathbf{V}}^\top\right)\right)^k$).

Mini-Batch. Recall that our optimization objective Eq. (5.11) includes two kernel matrices $\mathbf{K}_{\mathcal{TS}}$ and $\mathbf{K}_{\mathcal{SS}}$. The computation of every entry from $\mathbf{K}_{\mathcal{TS}}$ and the upper (or lower)-triangle of $\mathbf{K}_{\mathcal{SS}}$ is independent with each other. In this way, if the computation complexity of every entry from the kernel matrix is $O(C)$, the total complexity of computing $\mathbf{K}_{\mathcal{TS}}$ and $\mathbf{K}_{\mathcal{SS}}$ is $O\left(\left(n_{\mathcal{T}} n_{\mathcal{S}} + \frac{1}{2} n_{\mathcal{S}}^2\right) C\right)$, which are resource-intensive for some large graph datasets. Fortunately, KIDD and its variant are easy to mini-batch. Specifically, at every iteration, a subset of $\mathcal{T}$ and a subset of $\mathcal{S}$ are sampled, and this part of $\mathcal{S}$ is updated by minimizing Eq. (5.11) through gradient descent. This mini-batch technique is optional and can easily be incorporated with any of the above designs. Our efficiency study experiment shows that the mini-batch is especially important for our proposed KIDD-D and KIDD-LR.

Detailed Algorithms Detailed algorithms of KIDD-LR and KIDD-D are provided in Algorithm 5.1 and Algorithm 5.2, respectively.

Algorithm 5.1: KIDD-LR

Input : a target graph dataset $\mathcal{T} = \{(\mathcal{G}_i, y_i)\}_0^{n_{\mathcal{T}}-1}$, the size of the synthetic dataset $n_{\mathcal{S}}$;
Output: the synthetic dataset $\mathcal{S}$;

initialization: sample $n_{\mathcal{S}}$ graphs and their labels y_i from the $\mathcal{T}$ as the initial $\mathcal{S}$;
  decompose adjacency matrices into their low-rank matrices (e.g., $\tilde{\mathbf{U}}_i, \tilde{\mathbf{V}}_i$) by SVD.
while $\mathcal{S}$ *is not converged* **do**
  compute $\mathbf{K}_{\mathcal{TS}}$ and $\mathbf{K}_{\mathcal{SS}}$ by Eq. (5.14a), (5.14b), Eq. (5.17), and Eq. (5.14d);
  Update $\mathcal{S} = \{\tilde{\mathbf{U}}_i, \tilde{\mathbf{V}}_i, \tilde{\mathbf{X}}_i\}_0^{n_{\mathcal{S}}-1}$ based on the gradient $\frac{\partial \mathcal{L}}{\partial \mathcal{S}}$ from Eq. (5.11);
end
clip $\{\tilde{\mathbf{A}}_i = \tilde{\mathbf{U}}_i \tilde{\mathbf{V}}_i^\top\}_0^{n_{\mathcal{S}}-1}$ to be non-negative;
return $\mathcal{S} = \{\tilde{\mathbf{A}}_i, \tilde{\mathbf{X}}_i, y_i\}_0^{n_{\mathcal{S}}-1}$.

Algorithm 5.2: KiDD-D

Input : a target graph dataset $\mathcal{T} = \{(\mathcal{G}_i, y_i)\}_0^{n_\mathcal{T}-1}$, the size of the synthetic dataset $n_\mathcal{S}$;

Output: the synthetic dataset $\mathcal{S}$;

initialization: sample $n_\mathcal{S}$ graphs and their labels y_i from the $\mathcal{T}$ as the initial $\mathcal{S}$; initialize the Bernoulli parameter matrix $\tilde{\mathbf{B}}_i$ of every synthetic graph according to their adjacency matrix: $\tilde{\mathbf{B}}_i[u,v] = C$ if $\tilde{\mathbf{A}}_i[u,v] = 1$; otherwise $\tilde{\mathbf{B}}_i[u,v] = -C$, where C is a constant, e.g., 1.

while $\mathcal{S}$ *is not converged* **do**

  Sample adjacency matrices of synthetic graphs $\{\tilde{\mathbf{A}}_i\}_0^{n_\mathcal{S}-1}$ by Eq. (5.21) and $\{\tilde{\mathbf{B}}_i\}_0^{n_\mathcal{S}-1}$;

  compute $\mathbf{K}_{\mathcal{TS}}$ and $\mathbf{K}_{\mathcal{SS}}$ by Eq. (5.14a)-(5.14d);

  Update $\mathcal{S} = \{\tilde{\mathbf{B}}_i, \tilde{\mathbf{X}}_i\}_0^{n_\mathcal{S}-1}$ based on the gradient $\frac{\partial \mathcal{L}}{\partial \mathcal{S}}$ from Eq. (5.11);

end

Discretize the adjacency matrix of every synthetic graph: $\tilde{\mathbf{A}}_i[u,v] = 1$ if $\texttt{sigmoid}(\tilde{\mathbf{B}}_i[u,v]) > 0.5$, else $\tilde{\mathbf{A}}_i[u,v] = 0$;

return $\mathcal{S} = \{\tilde{\mathbf{A}}_i, \tilde{\mathbf{X}}_i, y_i\}_0^{n_\mathcal{S}-1}$.

5.1.4 Experiments

Table 5.1: Performance comparison (mean±std). The best and second-best results are bold and underlined, respectively.

Name	Graphs/Cls	Ratio	Random	Herding	K-Center	DosCond	KiDD-D	KiDD-LR	All data
NCI1 (ACC)	1	0.06%	57.4±3.0	59.2±3.0	59.2±3.0	57.1±0.9	60.1±0.9	**60.3±1.6**	80.0±1.1
	10	0.61%	59.9±2.0	62.8±0.9	59.1±0.8	60.8±0.9	61.7±1.3	**63.3±1.6**	
	50	3.04%	60.5±2.1	62.5±2.0	59.5±0.5	62.7±0.8	**64.2±0.6**	62.2±0.8	
NCI109 (ACC)	1	0.06%	54.3±2.3	51.7±0.9	51.7±0.9	54.9±2.3	**55.2±2.7**	54.4±1.0	77.7±0.6
	10	0.61%	61.9±1.6	**63.6±0.3**	52.9±1.7	61.4±1.5	63.5±0.5	62.8±0.7	
	50	3.03%	64.0±1.4	64.7±1.2	55.0±2.1	62.9±1.6	**70.4±1.3**	64.7±1.0	
PROTEINS (ACC)	1	0.22%	57.8±1.8	67.6±1.7	67.6±1.7	63.4±1.9	68.8±3.9	**69.3±4.9**	78.6±2.6
	10	2.25%	67.2±0.7	68.3±1.0	71.4±3.3	71.7±0.4	74.1±1.9	**75.3±0.4**	
	50	11.24%	69.6±4.0	70.1±1.0	72.9±2.6	73.2±0.8	75.0±1.9	**75.6±2.3**	
DD (ACC)	1	0.21%	61.3±8.5	60.7±8.4	61.0±3.2	63.0±0.7	65.8±1.7	**69.7±1.0**	76.9±3.2
	10	2.12%	66.8±2.1	67.4±0.7	66.2±2.4	68.1±1.8	70.6±1.3	**71.1±0.8**	
	50	10.62%	71.4±2.2	71.6±1.9	72.3±1.0	70.9±1.0	**73.1±2.2**	71.7±0.7	
molhiv (ROC-AUC)	1	<0.01%	0.555±0.036	0.633±0.025	0.633±0.025	0.612±0.025	0.633±0.016	**0.637±0.069**	0.750±0.007
	10	0.06%	0.579±0.012	0.621±0.009	0.630±0.013	0.647±0.031	**0.675±0.054**	0.654±0.019	
	50	0.30%	0.623±0.013	0.616±0.014	0.628±0.014	0.620±0.018	0.708±0.062	**0.709±0.020**	
molbbbp (ROC-AUC)	1	0.12%	0.579±0.025	**0.628±0.012**	**0.628±0.012**	0.584±0.030	**0.628±0.011**	0.623±0.006	0.650±0.014
	10	1.23%	0.556±0.003	0.625±0.002	0.596±0.016	0.621±0.013	**0.644±0.011**	0.631±0.015	
	50	6.13%	0.610±0.007	0.630±0.013	0.595±0.018	0.628±0.012	0.662±0.030	**0.663±0.016**	
molbace (ROC-AUC)	1	0.17%	0.638±0.009	0.546±0.038	0.546±0.038	0.667±0.021	0.693±0.016	**0.698±0.014**	0.727±0.017
	10	1.65%	0.649±0.017	0.561±0.041	0.658±0.016	0.694±0.018	**0.748±0.020**	0.717±0.012	
	50	8.26%	0.655±0.020	0.703±0.012	0.662±0.013	0.710±0.006	**0.766±0.007**	0.724±0.020	

Datasets. In this paper, we select 7 real-world graph classification datasets including NCI1, NCI109, DD, and PROTEINS from TUDataset [286] and ogbg-molhiv, ogbg-molbbbp, and ogbg-molbace from open graph benchmarks [30]. For NCI1, NCI109, DD, and PROTEINS, 80/10/10% of the graphs from every dataset are randomly split into the training/-validation/test set. For ogbg-molhiv, ogbg-molbbbp, and ogbg-molbace, we use their default dataset split. The detailed dataset statistics are provided in Table 5.2.

Table 5.2: Dataset statistics.

Name	**# Graphs**	**# Nodes**	**# Edges**	**# Features**	**# Classes**
NCI1	4110	29.9	32.3	37	2
NCI109	4127	29.7	32.1	38	2
PROTEINS	1113	39.1	72.8	4	2
DD	1178	284.3	715.7	89	2
ogbg-molhiv	41127	25.5	54.9	9	2
ogbg-molbbbp	2039	24.1	51.9	9	2
ogbg-molbace	1513	34.1	73.7	9	2

Metrics. We select GIN [299] as the downstream GNN, which is trained on the distilled graph dataset $\mathcal{S}$. Its performance on the test graphs is the metric of the corresponding distilled training graphs. To be specific, for NCI1, NCI109, DD, and PROTEINS, accuracy (ACC) is reported and for ogbg-molhiv, ogbg-molbbbp, and ogbg-molbace ROC-AUC is reported as the datasets suggested. We report the average result and the standard deviation in 10 runs.

Baseline Methods. We select 4 baseline methods including 3 core-set methods (Random, Herding [307], and K-Center [308, 309]) and a graph dataset distillation method DosCond [45]. Concretely, Random selection is the most naive method which randomly samples $\mathcal{S}$ from $\mathcal{T}$. For Herding and K-Center, we first learn the representation of every training graph by the GIN [299] trained on the whole training set. Then, Herding selects the closest samples to the cluster center of every class. K-Center selects the center samples such that the distance between every node to its nearest center is minimized. Specifically, we implement a greedy solution of K-Center [310] whose initialized set is from Herding. DosCond is a learning-based graph dataset distillation method that matches the training gradient on $\mathcal{S}$ and $\mathcal{T}$ at the initialization step. DosCond is fast, but unlike the exact solution adopted in KIDD, it applies bold approximations for the bi-level distillation objective function.

Implementation of KiDD. In our implementation of KIDD, LiteGNTK is applied to all the graph kernel computations. KIDD-LR is the variant that uses the proposed low-rank speed-up designs, and KIDD-D is the variant that uses the Gumbel-Max trick to synthesize discrete graph topology. Mini batch is flexibly applied depending on the size of the target dataset. The code is provided[1].

Effectiveness of KiDD-LR and KiDD-D. For every dataset, 1/10/50 graphs are distilled for every class by baselines and our methods, respectively. After that, the downstream graph classifier GIN [299] is trained on the distilled training graphs, and we report its performance on the test graphs in Table 5.1. The rightmost column shows the test performances of the downstream classifiers trained on the entire original training sets. The effectiveness comparison is provided in Table 5.1. It is observed that

- In most cases, as expected, with the increasing number of training graphs, the downstream graph classifier's performances are improved. This observation is consistent among both the coreset methods (Random, Herding, K-Center) and learning-based distillation methods (DosCond, KIDD-D, and KIDD-LR).
- Interestingly, in some cases (e.g., DD), Random is not always the weakest baseline method, even though it is the most naive one. It reflects that the most representative training samples could be hard to find by heuristics (e.g., Herding, K-Center) and shows the advantages of the learning-based methods (DosCond, KIDD-D, and KIDD-LR).
- The proposed KIDD-D and KIDD-LR obtain the best performance against all the baseline methods under most settings. Strikingly, on ogbg-molbbbp and ogbg-molbace datasets, when the numbers of synthetic graphs are only 1.65%-8.26% of the training graphs, the graph classifiers trained on such tiny datasets are able to outperform the correspondences trained on the complete datasets. It further demonstrates the advantage of learning a representative and informative training set over directly sampling the representative ones from the existing training graphs.

Efficiency Study. We first verify the efficiency improvement of our proposed enhanced designs. Specifically, we measure the wall clock time during the forward computation and backward gradient propagation (implemented with PyTorch). 3 models are compared: (1) KIDD-GNTK: using GNTK as the kernel but not our proposed LiteGNTK; (2) KIDD-LiteGNTK: using LiteGNTK as the graph kernel; (3) KIDD-LiteGNTK-LR: using LiteGNTK as the graph kernel and applying the low-rank speed-up technique whose rank is set

[1] https://github.com/pricexu/KIDD

Table 5.3: Efficiency comparison (second/iteration) of KIDD with different kernels. $B_{\mathcal{T}}, B_{\mathcal{S}}$ are the batch sizes of the target and synthetic training sets, respectively.

$(B_{\mathcal{T}}, B_{\mathcal{S}})$	Computation	GNTK	LiteGNTK	LiteGNTK-LR
(32, 2)	Forward	0.0475	0.0153	0.0071
	Backprop	0.0241	0.0078	0.0040
(64, 2)	Forward	0.0697	0.0212	0.0111
	Backprop	0.0251	0.0094	0.0045
(128, 16)	Forward	0.2004	0.0551	0.0288
	Backprop	0.1525	0.0433	0.0173
(256, 32)	Forward	OOM	0.1341	0.0745
	Backprop	OOM	0.1450	0.0633

as 16 in this experiment. All the graph neural network kernels have 5 aggregate operations. For KIDD-GNTK and KIDD-LiteGNTK, they do not include the Gumbel-Max reparameterization trick because it is not proposed for better training efficiency, and in addition, it is an element-wise operation (i.e., $\tilde{\mathbf{A}}[u,v] = \texttt{Gumbel-Max}\left(\tilde{\mathbf{B}}[u,v]\right)$) whose computation is not heavy. In this experiment, the dataset PROTEINS is used. We test four settings with different batch sizes of the target training graphs $\mathcal{T}$ and the synthetic graphs $\mathcal{S}$. The wall clock time comparison is presented in Table 5.3 from which we observe that

- KIDD-LiteGNTK is significantly faster (3× faster) than KIDD-GNTK as expected because the non-linearity between aggregation operations is removed, and the aggregation operations (i.e., $\mathbf{A}^k$) can be precomputed. Also, as GNTK contains more operations involving more intermediate variables for modern machine learning packages (e.g., PyTorch) that leads to heavier memory usage. E.g., for the case with batch size $(256, 32)$, KIDD-GNTK is out of memory.

- KIDD-LiteGNTK-LR is much faster than KIDD-LiteGNTK, which aligns well with the Lemma 5.1.

Next, we compare the wall clock time of KIDD-D and KIDD-LR compared with the learning-based graph dataset distillation method DosCond, which applies a fast approximation of the bilevel gradient matching loss. As the forward and backward computation of DosCond is not clearly defined, whose forward computation of the gradient matching loss involves a gradient backpropagation, we record and compare the total training time for a fixed number of iterations. As the official implementation of Doscond [2] applies mini batch towards the target training set $\mathcal{T}$ while updating $\mathcal{S}$ in a full-batch fashion. For a fair comparison, we follow their settings and set all the methods' batch size of $\mathcal{T}$ as 64, the batch

[2]https://github.com/amazon-science/doscond

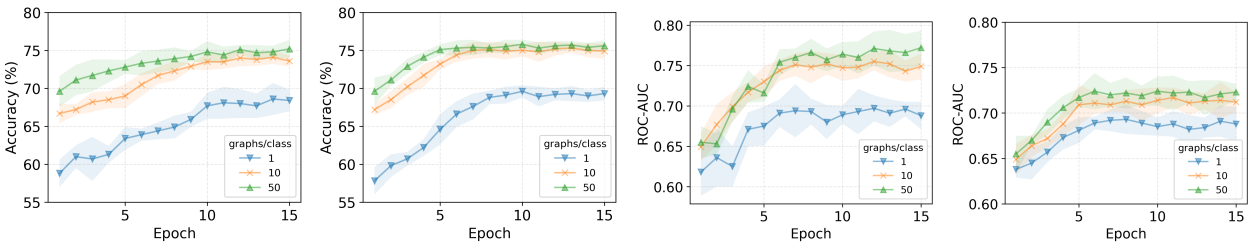
Table 5.4: Efficiency comparison (second/iteration) with the baseline method DosCond.

Dataset	Method	Graphs/Class		
		1	**10**	**50**
NCI1	DosCond	0.6755	0.6804	0.7009
	KIDD-D	0.0255	0.0463	0.1540
	KIDD-LR	0.0122	0.0240	0.0826
PROTEINS	DosCond	0.2027	0.2077	0.2112
	KIDD-D	0.0323	0.0608	0.2095
	KIDD-LR	0.0157	0.0313	0.1090
ogbg-molhiv	DosCond	0.0274	0.0311	0.0328
	KIDD-D	0.0281	0.0492	0.1486
	KIDD-LR	0.0135	0.0250	0.0748

size of $\mathcal{S}$ is the # classes $\times$ graphs/class, and both methods' number of the aggregation layers as 3. Then, the wall clock time per update iteration is recorded. The wall clock time comparison is presented in Table 5.4, from which we observe

- The efficiency of DosCond is consistent with respect to the batch size of the synthetic graphs $\mathcal{S}$. As a comparison, The efficiency of our method KIDD-LR and KIDD-D is more sensitive to the batch size of the synthetic graphs. That is because DosCond's time complexity is linear with respect to $B_{\mathcal{T}}+B_{\mathcal{S}}$ but our method needs to compute two kernel matrices $\mathbf{K}_{\mathcal{TS}}$ and $\mathbf{K}_{\mathcal{SS}}$ whose numbers of entries are $B_{\mathcal{T}}B_{\mathcal{S}}$ and $B_{\mathcal{S}}^2$, respectively. Here $B_{\mathcal{T}}$, $B_{\mathcal{S}}$ are the batch size of $\mathcal{T}$ and $\mathcal{S}$, respectively.
- As we claimed, both KIDD-D and KIDD-LR cannot scale to very large batch sizes (e.g., 1024). However, with an appropriate batch size (e.g., $B_{\mathcal{T}} = 64$ and $B_{\mathcal{S}} = 20$), two KIDD variants, especially the KIDD-D can have comparable or even better efficiency than DosCond. This is surprising as the DosCond solves the bi-level optimization objective approximately, but KIDD-D provides an exact solution for the distillation optimization objective.

Convergence Study. As a supplement to the efficiency study, a convergence study is provided which shows the models' performance with respect to the increase of training epochs. Here we select the PROTEINS and ogbg-molbace datasets and present the accuracy/ROC-AUC of our proposed models KIDD-D and KIDD-LR in Figure 5.1a-5.1d which shows our models can converge quickly within 15 epochs. Notice that in this experiment we select the batch size of $\mathcal{T}$ as 256 indicating on the PROTEINS dataset, 1 epoch is equivalent to



(a) KIDD-D on PROTEINS (b) KIDD-LR on PROTEINS (c) KIDD-D on ogbg-molbace (d) KIDD-LR on ogbg-molbace

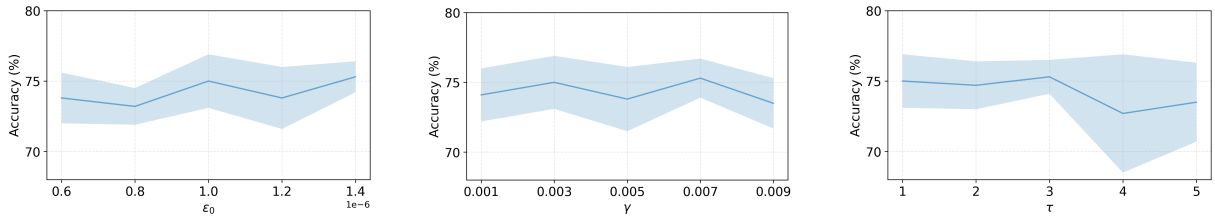
Figure 5.1: Model performance vs. training epochs.

4 update iterations, and on the ogbg-molbace dataset, 1 epoch is equivalent to 7 update iterations. As expected, the training of KIDD-D will be more unstable compared with the training of KIDD-LR due to the sampling operation in the forward computation.

Ablation Study. We conduct an ablation study to evaluate the effectiveness of each module in our proposed methods. Specifically, the following models are tested: (1) KIDD-D and KIDD-LR: two variants of our proposed KIDD; (2) KIDD-RWK: using random walk graph kernel (i.e., Eq. (5.14a), (5.14c), and (5.14d)) to compute the kernel matrix; (3) KIDD-D-NR and KIDD-LR-NR: removing the regularization term (i.e., Eq. (5.12)) from the optimization objective. Datasets PROTEINS, ogbg-molbbbp, and ogbg-molbace are selected, and the graphs/class is set as 50. The results are presented in Table 5.5, from which we observe

- The performance of KIDD-RWK is significantly lower than other KIDD variants equipped with LiteGNTK. It suggests that the random walk graph kernel (RWK)-based ridge regression cannot provide generalizable distilled graph datasets for a GNN classifier.
- The regularization term (i.e., Eq. (5.12)) can improve the performance of KIDD. Including this term in the training loss, our proposed KIDD-LR and KIDD-D obtain the best performances.

Sensitivity Study. There are three main hyperparameters of the proposed KIDD model, ϵ from Eq. (5.11), γ from Eq. (5.13), and τ from Eq. (5.21). In this study, we conduct experiments on the PROTEINS dataset with 50 synthetic graphs per class. The KIDD-D variant is tested because the hyperparameter τ is only used for the discrete scenarios. As we mentioned in the experimental settings above, we set $\epsilon = \epsilon_0 \times \frac{\texttt{trace}(\mathbf{K}_{\mathcal{SS}})}{n_{\mathcal{S}}}$ so that ϵ is stable with respect to the number of synthetic graphs. Then, we present the model's



(a) Accuracy (%) vs. varying ϵ_0 (b) Accuracy (%) vs. varying γ (c) Accuracy (%) vs. varying initial τ

Figure 5.2: Hyperparameter sensitivity studies.

Table 5.5: Ablation study results (mean±std). The metric for the PROTEINS is accuracy, and the metric for the ogbg-molbbbp and ogbg-molbace is ROC-AUC. The best is bold.

Method	PROTEINS	ogbg-molbbbp	ogbg-molbace
KIDD-RWK	64.8±2.4	0.578±0.009	0.629±0.013
KIDD-LR-NR	72.2±2.8	0.645±0.005	0.699±0.026
KIDD-D-NR	72.0±1.5	0.634±0.030	0.713±0.013
KIDD-LR	**75.6±2.3**	**0.663±0.016**	0.724±0.020
KIDD-D	75.0±1.9	0.662±0.030	**0.766±0.007**

performance with varying ϵ_0 in Figure 5.2a. For the different settings of hyperparameter γ, KIDD-D's performance is reported in Figure 5.2b. For the hyperparameter τ, as mentioned in the hyperparameter settings, we follow the suggestions from [45, 311] and anneal the initial τ to 0.01τ during the training process. Here we compare the performance of KIDD-D with varying initial τ as shown in Figure 5.2c. In general, we observe that KIDD-D's performance is stable with respect to the selection of hyperparameters ϵ_0 and γ. As τ increases, the model's performance becomes more and more unstable since large τ will lead to large gradients [294] and thus affect the training stability.

Hyperparameter Settings. The parameters of the KIDD-D and KIDD-LR are set as follows. The node scaling factor c_u is set as 1 for every node u from $\mathcal{T}$ and $\mathcal{S}$. The learning rate of KIDD-D and KIDD-LR is searched in $\{1e-1, 1e-2, 1e-3\}$. The ϵ is set as $\epsilon = \epsilon_0 \times \frac{\texttt{trace}(\mathbf{K}_{\mathcal{SS}})}{n_{\mathcal{S}}}$ so that it is stable with respect to the size of $\mathbf{K}_{\mathcal{SS}}$ and the ϵ_0 is set as $1e-6$. γ is searched in $\{0, 1e-4, 1e-3, 1e-2\}$. For KIDD-D, the τ is annealed during the training as [311] suggested. The initial τ is set as 1 and annealed to 0.01 after epoch 100. The rank r of the KIDD-LR is searched in $\{16, 32\}$. We plan to release the code upon publication.

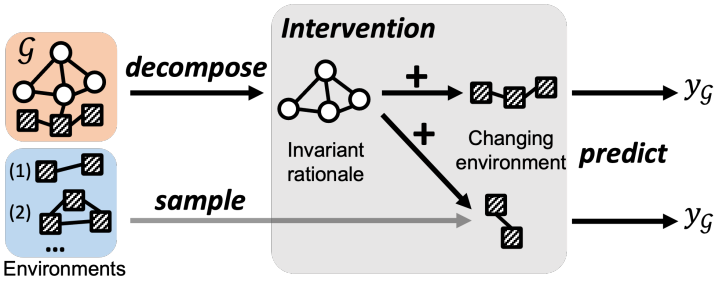


Figure 5.3: Illustration of the rationale/environment decomposition and intervention. Round nodes denote graph rationales, and square nodes (with stripes) denote the environments. The intervention aims to ensure the rationale from graph $\mathcal{G}$ truly has the discriminative power for the label $y_{\mathcal{G}}$.

5.2.1 Introduction

Rationale refers to a subset of the input features that play a crucial role in model predictions for downstream tasks [312, 313, 314, 315, 316, 317]. In the context of graph machine learning, graph rationale is defined as a subgraph of the input graph containing the most task-relevant semantics.

The application of graph rationale is broad, for example, it can greatly enhance model performance for graph-level tasks [314] by identifying the key components of the input graph. Additionally, the discovery of rationales can improve model explainability [313], as it highlights the parts of the input graph that significantly contribute to the final prediction.

Existing graph rationalization solutions [313, 314] employ a trainable *augmenter* to execute the rationale/environment decomposition. In this process, a node/edge mask is generated by the augmenter to decompose the given graph into a rationale graph and an environment graph. Inspired by the content-style decomposition [318], the key idea of graph rationalization is to preserve the utility of the graph rationale even when faced with changing environment graphs (see Figure 5.3). To achieve this, a technique named *intervention* is used, where the environment graph interacts with the rationale graph.

The intervention mechanism (named *intervener*) is essential in the graph rationalization process, as it must accurately represent the interaction between the rationale and the environment. Intuitively, the intervener should work in an adversarial manner against the augmenter mentioned above, a point not emphasized in the existing literature. If the intervener is more powerful, it can capture more detailed interactions between the rationale and

environment subgraphs. Given such a powerful intervener, the augmenter is compelled to minimize these interactions between the graph rationale and the environment to obtain a "purer" graph rationale.

Unfortunately, existing works develop interveners in a coarse and non-parametric manner. After performing rationale/environment decomposition on the graph data, they compute graph-level embeddings for the rationale and environment subgraphs. The intervention is then designed as an interaction between these graph-level embeddings. For example, [313] adds the environment embedding into the rationale embedding as the intervened rationale embedding; [314] defines the intervened prediction as the Hadamard product between the predictions based on the rationale subgraph and the environment subgraph. We argue that such a graph-level non-parametric intervention is insufficient to represent the interaction between the rationale and environment graphs effectively.

In response to this limitation, we propose a fine-grained, parametric intervention mechanism named FIne-grained Graph rationalization (FIG) [47]. Our proposed FIG draws inspiration from the self-attention module in the Transformer model, which captures interactions between input tokens. Building upon insights from Transformer [319, 320] and its linear variant Linformer [321], FIG formulates the interaction between the rationale and environment subgraphs at the node-level or the virtual node-level. The two variants are named FIG-N and FIG-VN. Additionally, to maximize the effectiveness of the intervention, we formulate a min-max game involving the node encoder, augmenter, intervener, and predictor, compelling the rationale subgraph to be as informative as possible.

We conduct comprehensive experiments on 7 graph-level benchmarks to evaluate the proposed approach and compare FIG-N/VN against 13 state-of-the-art baseline methods. The results demonstrate that our proposed FIG and its variants outperform the baseline methods, validating their superior performance.

5.2.2 Preliminaries

Notations. We adopt the following notation conventions: bold uppercase letters for matrices and tensors (e.g., $\mathbf{A}$), bold lowercase letters for column vectors (e.g., $\mathbf{u}$), lowercase and uppercase letters in regular font for scalars (e.g., d, K), and calligraphic letters for sets (e.g., $\mathcal{T}$). To index vectors/matrices/tensors, we follow the syntax from NumPy[3] (0-based). Specifically, $\mathbf{A}[p,:]$ and $\mathbf{A}[:,q]$ represent the p-th row and the q-th column of matrix $\mathbf{A}$ respectively; $\mathbf{A}[p,q]$ represents the entry at the p-th row and the q-th column. Similarly,

[3] `https://numpy.org/doc/stable/index.html`

$\mathbf{u}[p]$ denotes the p-th entry of vector $\mathbf{u}$. In addition, the slicing syntax for vectors/matrices/tensors is used. For example, for a matrix $\mathbf{A}$, $\mathbf{A}[i:j,:]$ denotes rows from the i-th row (included) to the j-th row (excluded) and $\mathbf{A}[:,:k]$ denotes all the columns before the k-th column. The superscript $\top$ denotes the transpose of matrices and vectors. $\odot$ represents the Hadamard product, and $\circ$ denotes function composition. We use $||$ to represent the concatenation operation, and the specific dimension of concatenation will be clarified based on the context.

An attributed graph can be represented as $\mathcal{G} = (\mathbf{A}, \mathbf{X}, \mathbf{E})$, where $\mathbf{A} \in \mathbb{R}^{n \times n}$ is the adjacency matrix, $\mathbf{X} \in \mathbb{R}^{n \times d_X}$ is the node feature matrix, and $\mathbf{E} \in \mathbb{R}^{n \times n \times d_E}$ is the edge feature tensor. Here, n denotes the number of nodes, and d_X (or d_E) represents the dimensions of node (or edge) features, respectively. This paper assumes the node and edge feature dimensions are the same (i.e., $d_X = d_E = d$) for brevity; if they differ, a simple, fully connected layer can map them into a common feature space. Our main focus in this paper is on graph property prediction tasks. The ground truth of a graph is represented by y.

Graph Transformer. The core modules of the Transformer architecture [319] are the self-attention layer and the feed-forward network layer. Given the input as a sequence of symbol representations $\mathbf{H} \in \mathbb{R}^{n \times d_H}$, it is first transformed into the query, key, and value matrices as

$$\mathbf{Q} = \mathbf{H}\mathbf{W}_Q, \mathbf{K} = \mathbf{H}\mathbf{W}_K, \mathbf{V} = \mathbf{H}\mathbf{W}_V, \tag{5.22}$$

where $\mathbf{W}_Q \in \mathbb{R}^{d_H \times d_Q}$, $\mathbf{W}_K \in \mathbb{R}^{d_H \times d_K}$, $\mathbf{W}_V \in \mathbb{R}^{d_H \times d_V}$. For the brevity of the presentation, we set $d_H = d_Q = d_K = d_V = d$. Then, the self-attention module works as,

$$\mathbf{P} = \texttt{attn}(\mathbf{H}) = \sigma(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}), \tag{5.23a}$$

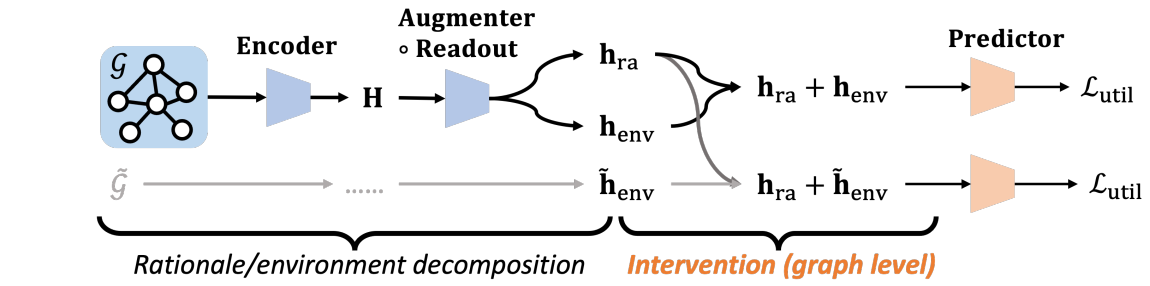
$$\mathbf{H} \leftarrow \mathbf{P}\mathbf{V} + \mathbf{H}. \tag{5.23b}$$

Typically, the non-linearity σ is `softmax`. The feed-forward network (FFN) updates the symbol representations $\mathbf{H}$ as:

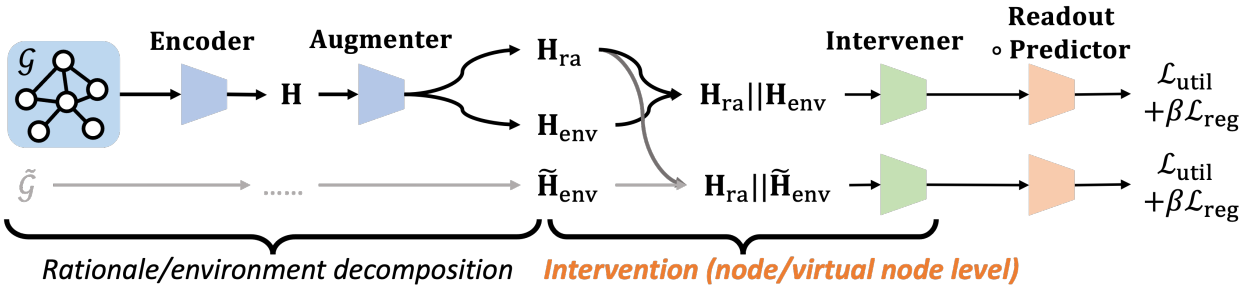
$$\mathbf{H} \leftarrow \texttt{FFN}(\mathbf{H}) + \mathbf{H}. \tag{5.24}$$

Additional techniques such as layer/batch normalization [322, 323], dropout [324], and multi-head attention [319] can be included, but omitted here for brevity.

While Transformers was initially devised for sequence or set data with positional encoding, numerous techniques have since been introduced to adapt Transformers for graph data. Based on the taxonomy outlined by [201], most graph Transformers are designed from the



(a) GREA [313]



(b) FIG (Ours)

Figure 5.4: Pipeline comparison between existing work, GREA, and proposed FIG. ∘ denotes function composition. GREA designs the intervention at the graph level, while the proposed FIG designs the intervention at the node or virtual node level. The augmented environment $\tilde{\mathbf{H}}_{\texttt{env}}$ is from another graph $\tilde{\mathcal{G}}$ (through the Encoder and Augmenter) in the batch.

perspectives of (1) incorporating the topology encoding into the node features, (2) incorporating the topology encoding into the attention matrix, and (3) utilizing graph neural networks [325] as auxiliary modules.

Interestingly, it is well-known in both the graph learning [326] and natural language processing communities [321, 327] that, from the message-passing perspective, the key idea of the Transformer architecture is to reconstruct a weighted complete graph, whose adjacency matrix is $\mathbf{P} = \sigma\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\right)$.

Invariant Rationale Discovery on Graphs. The graph rationale is a subgraph that encodes most downstream task-relevant semantics. A typical example is the functional groups in polymer graphs [313, 314], which fundamentally determines the chemical property of polymers. Mathematically, a given graph is decomposed into a rationale graph and an environment graph: $\mathcal{G} = \mathcal{G}_{\texttt{ra}} \cup \mathcal{G}_{\texttt{env}}$. Commonly, the graph embeddings on $\mathcal{G}_{\texttt{ra}}$ and $\mathcal{G}_{\texttt{env}}$ are computed as $\mathbf{h}_{\texttt{ra}}$ and $\mathbf{h}_{\texttt{env}}$. To ensure the rationale graph is invariant w.r.t. the prediction results when confronting different environments, a utility loss is minimized given the rationale embedding $\mathbf{h}_{\texttt{ra}}$ intervened by the environment embedding $\tilde{\mathbf{h}}_{\texttt{env}}$, i.e., $\min \mathcal{L}_{\texttt{util}}(\mathbf{h}_{\texttt{ra}} \xleftarrow{\text{intervene}}$

$\tilde{\mathbf{h}}_{\texttt{env}}$). Here, $\tilde{\mathbf{h}}_{\texttt{env}}$ could be either from the same graph (i.e., $\tilde{\mathbf{h}}_{\texttt{env}} = \mathbf{h}_{\texttt{env}}$), or could be environment embeddings from other graphs, such as those in the batch. A key difference among existing methods lies in the intervention operation, of which we mention two:

- GREA [313] designs the intervention as adding operation, i.e., $\mathbf{h}_{\texttt{ra}} + \tilde{\mathbf{h}}_{\texttt{env}}$;
- DIR [314] designs the intervention as an element-wisely weighting of prediction: $\theta_{\texttt{pred}}(\mathbf{h}_{\texttt{ra}}) \odot \texttt{sigmoid}(\theta_{\texttt{pred}}(\tilde{\mathbf{h}}_{\texttt{env}}))$, where $\odot$ is the Hadamard product and $\theta_{\texttt{pred}}$ is a predictor.

The above intervention is conducted at the graph level because $\mathbf{h}_{\texttt{ra}}$ and $\tilde{\mathbf{f}}_{\texttt{env}}$ are graph embeddings. In Figure 5.4a, an overview of the GREA [313] is presented. For comparison, we aim to design the intervention at a finer grain (e.g., node level) to handle the interaction between the rationale and environment graphs, which will be detailed as follows.

5.2.3 Proposed Model: Encoder and Predictor

In this section, we introduce our proposed graph rationalization method, FIG. At its core, FIG utilizes a module based on the Transformer architecture. Figure 5.4b provides an overview of FIG, highlighting its four main parametric modules: the encoder, augmenter, intervener, and predictor.

Encoder. The encoder, denoted as $\theta_{\texttt{enc}} : \mathcal{G} \to \mathbb{R}^{n \times d}$, accepts a graph data as input and produces a node embedding matrix as output. While there are various graph encoders available, such as graph neural networks (GNNs) [325] and graph Transformers [201]. From the methodology perspective, the encoder module is not the main contribution of this paper, so in this section, we do not specify a specific graph encoder $\theta_{\texttt{enc}}$.

Predictor. The predictor, denoted as $\theta_{\texttt{pred}} : \mathbb{R}^d \to \mathbb{R}^c$ takes as input a graph embedding and outputs a task-related vector/scalar. For graph regression tasks, $c = 1$; for graph classification tasks, c is the number of classes. A typical choice of predictor is a multi-layer perceptron (`MLP`) with appropriate activation functions. Details of the encoder and predictor in our implementation are presented in Section 5.2.6.

In subsequent subsections, we will elaborate on the augmenter and intervener, two essential modules. Their detailed designs derive two variants of the proposed FIG model.

5.2.4 Node-Level Variant: FIG-N

Node-Level Augmenter. The augmenter is a critical module of the proposed FIG. For the node-level variant, termed FIG-N, the augmenter's primary function is to decompose the node set into two distinct subsets: rationale nodes and environment nodes. This decomposition is operated by parameterizing the node-level augmenter as a learnable node partitioner, denoted by $\theta_{\texttt{aug-N}}$,

$$\mathbf{m} = \texttt{sigmoid}(\texttt{MLP}(\mathbf{H}, \theta_{\texttt{aug-N}})), \tag{5.25}$$

whose input is the node embedding matrix $\mathbf{H} \in \mathbb{R}^{n \times d}$, and its output is a partition vector $\mathbf{m} \in [0,1]^n$. `MLP` is a multi-layer perceptron. Each entry within $\mathbf{m}$, such as $\mathbf{m}[i]$, signifies the probability of the i-th node being categorized as a rationale node.

For the node partition vector $\mathbf{m}$, its top-K entries are indexed as $\texttt{idx}_{\texttt{ra}} = \texttt{argtopK}(\mathbf{m})$ which is used to index the rationale nodes from the node embedding matrix $\mathbf{H}$; naturally, the remaining nodes are categorized as the environment nodes whose indices are $\texttt{idx}_{\texttt{env}} = \{1, \ldots, n\} - \texttt{idx}_{\texttt{ra}}$. K is a hyper-parameter whose impact is studied in Section 5.2.6. Also, in our implementation, we use a soft `argtopK` operation to remain differentiability whose details are in Section 5.2.7.

Using the indices mentioned above, rationale and environment embeddings, denoted as $\mathbf{H}_{\texttt{ra}}$ and $\mathbf{H}_{\texttt{env}}$, respectively, can be extracted from the node embedding matrix $\mathbf{H}$:

$$\mathbf{H}_{\texttt{ra}} = \mathbf{H}[\texttt{idx}_{\texttt{ra}}, :] \in \mathbb{R}^{K \times d}, \tag{5.26a}$$

$$\mathbf{H}_{\texttt{env}} = \mathbf{H}[\texttt{idx}_{\texttt{env}}, :] \in \mathbb{R}^{(n-K) \times d}, \tag{5.26b}$$

Node-Level Intervener. The design of the fine-grained intervener draws inspiration from the Transformer architecture [319]. Explicitly, the node-level intervener ϕ is presented as,

$$\mathbf{H}_{\texttt{inter}}, \mathbf{P} = \texttt{transformer}(\mathbf{H}_{\texttt{ra}} || \mathbf{H}_{\texttt{env}}), \tag{5.27a}$$

$$\textit{where} \quad \mathbf{P} = \texttt{attn}(\mathbf{H}_{\texttt{ra}} || \mathbf{H}_{\texttt{env}}). \tag{5.27b}$$

In this representation, the operator $||$ concatenates along the first dimension of the matrices $\mathbf{H}_{\texttt{ra}}$ and $\mathbf{H}_{\texttt{env}}$. We dub the Eqs. (5.22)-(5.24) as `transformer` and $\mathbf{P}$ is the intermediate attention matrix from the self-attention layer (Eq. (5.27b)). Here, the self-attention module models the interactions between the rational nodes $\mathbf{H}_{\texttt{ra}}$ and the environment nodes $\mathbf{H}_{\texttt{env}}$. ϕ includes all the parameters of the `attn` (Eq. (5.27b)) and `FFN` (Eq. (5.24)) modules. In some contexts where the attention matrix $\mathbf{P}$ is not explicitly used as an output, input/output of the intervener ϕ can be presented as $\mathbf{H}_{\texttt{inter}} = \phi(\mathbf{H}_{\texttt{ra}} || \mathbf{H}_{\texttt{env}})$.

FIG-N Optimization Objective. The utility loss is computed as $\mathcal{L}_{\texttt{util}}(\mathbf{H}_{\texttt{ra}}||\mathbf{H}_{\texttt{env}}) = \mathcal{L}_{\texttt{task}}(\theta_{\texttt{pred}} \circ \texttt{Readout} \circ \phi(\mathbf{H}_{\texttt{ra}}||\mathbf{H}_{\texttt{env}}), \mathbf{y})$, where $\mathcal{L}_{\texttt{task}}$ is the task-specific objective. For instance, it could be the mean squared error for regression tasks or the cross-entropy for classification tasks. As introduced in Figure 5.3, the core of the invariant rationale discovery is to find the graph rationale so that the utility loss attains minimization given *changing environments*. Thus, the total utility objective is

$$\mathcal{L}_{\texttt{util}} = \mathcal{L}_{\texttt{util}}\left(\mathbf{H}_{\texttt{ra}}||\mathbf{H}_{\texttt{env}}\right) + \alpha\mathcal{L}_{\texttt{util}}\left(\mathbf{H}_{\texttt{ra}}||\tilde{\mathbf{H}}_{\texttt{env}}\right), \tag{5.28}$$

where $\tilde{\mathbf{H}}_{\texttt{env}}$ is the node embeddings from the changing environments. In practical implementations, $\tilde{\mathbf{H}}_{\texttt{env}}$ is the environment node embeddings from other graphs in the mini-batch. Additionally, to fully utilize the rich interactions from the fine-grained intervention module, we apply the following partition regularization term,

$$\mathcal{L}_{\texttt{reg}}(\mathbf{H}_{\texttt{ra}}||\mathbf{H}_{\texttt{env}}) = \mathbf{s}^\top\mathbf{P}(\mathbf{1}-\mathbf{s}) + (\mathbf{1}-\mathbf{s})^\top\mathbf{P}\mathbf{s}, \tag{5.29}$$

where $\mathbf{P} \in \mathbb{R}^{n\times n}$ is the self-attention matrix from Eq. (5.27b),

$$\mathbf{s}[i] = \begin{cases} 1 & \text{if } i < K. \\ 0 & \text{otherwise.} \end{cases} \tag{5.30}$$

0-based indexing is used so there are in total K non-zero entries (i.e., 1) in $\mathbf{s}$. The meaning of the binary $\mathbf{s}$ vector is to designate whether a particular row of the matrix $\mathbf{H}_{\texttt{ra}}||\mathbf{H}_{\texttt{env}}$ originates from the rationale nodes or the environment nodes. The underlying notion of the regularization term Eq. (5.29) is to impose penalties on interactions between the rationale nodes and the environment nodes. Namely, these two terms $\mathbf{s}^\top\mathbf{P}(\mathbf{1}-\mathbf{s})$ and $(\mathbf{1}-\mathbf{s})^\top\mathbf{P}\mathbf{s}$ denote the total weights on the links (i.e., cut) between the rationale and environment subgraphs. To handle the changing environments, we introduce an additional regularization term on the changing environments as $\mathcal{L}_{\texttt{reg}}(\mathbf{H}_{\texttt{ra}}||\tilde{\mathbf{H}}_{\texttt{env}})$ where $\tilde{\mathbf{H}}_{\texttt{env}}$ is the environment node embeddings from another graph within the same mini-batch. Then, the total regularization term is

$$\mathcal{L}_{\texttt{reg}} = \mathcal{L}_{\texttt{reg}}\left(\mathbf{H}_{\texttt{ra}}||\mathbf{H}_{\texttt{env}}\right) + \mathcal{L}_{\texttt{reg}}\left(\mathbf{H}_{\texttt{ra}}||\tilde{\mathbf{H}}_{\texttt{env}}\right), \tag{5.31}$$

and the total objective function is $\mathcal{L}_{\texttt{util}} + \beta\mathcal{L}_{\texttt{reg}}$. To fully harness the capabilities of the fine-grained parametric intervener, it is crucial to noteas highlighted in the introductionthat the behavior of the intervener ϕ operates in an adversarial fashion to the other modules. As

Algorithm 5.3: FIG-N single training step for every training graph $\mathcal{G}$

Input : a labelled graph $(\mathcal{G}, y)$, a sampled graph $\tilde{\mathcal{G}}$ from the same batch as $\mathcal{G}$, $\theta = \{\theta_{\texttt{enc}}, \theta_{\texttt{aug-N}}, \theta_{\texttt{pred}}\}$, ϕ;

Output: updated θ and ϕ;

compute $\mathbf{H} = \theta_{\texttt{enc}}(G)$ and $\tilde{\mathbf{H}} = \theta_{\texttt{enc}}(\tilde{G})$;

compute $(\mathbf{H}_{\texttt{ra}}, \mathbf{H}_{\texttt{env}}) = \theta_{\texttt{aug-N}}(\mathbf{H})$, $(\tilde{\mathbf{H}}_{\texttt{ra}}, \tilde{\mathbf{H}}_{\texttt{env}}) = \theta_{\texttt{aug-N}}(\tilde{\mathbf{H}})$ via Eqs. (5.25), (5.26a), and (5.26b);

concatenate rationale-environment pairs $\mathbf{H}_{\texttt{ra}}||\mathbf{H}_{\texttt{env}}$ and $\mathbf{H}_{\texttt{ra}}||\tilde{\mathbf{H}}_{\texttt{env}}$;

compute $\mathcal{L}_{\texttt{util}}$ via Eq. (5.28);

compute $\mathcal{L}_{\texttt{reg}}$ via Eqs. (5.31) and (5.30);

update θ via gradient descent with $\frac{\partial(\mathcal{L}_{\texttt{util}}+\beta\mathcal{L}_{\texttt{reg}})}{\partial\theta}$;

update ϕ via gradient ascent with $\frac{\partial(\mathcal{L}_{\texttt{util}}+\beta\mathcal{L}_{\texttt{reg}})}{\partial\phi}$;

a result, we formulate a min-max game that involves $\theta = \{\theta_{\texttt{enc}}, \theta_{\texttt{aug-N}}, \theta_{\texttt{pred}}\}$ and ϕ as,

$$\min_{\theta} \max_{\phi} \quad \mathcal{L}_{\texttt{util}} + \beta\mathcal{L}_{\texttt{reg}}. \tag{5.32}$$

Here, the intervener ϕ is trained to decrease the utility of the graph rationale by promoting interactions between the rationale nodes and the environment nodes. Conversely, the encoder, augmenter, and predictor (i.e., θ) are optimized in an opposing manner to the intervener's objectives.

Complexity of FIG-N. As the encoder $\theta_{\texttt{enc}}$ and the predictor $\theta_{\texttt{pred}}$ are off-the-shelf, the FIG-N introduces two new modules: the node-level augmenter, $\theta_{\texttt{aug-N}}$, and the Transformer-based intervener, ϕ. Notably, despite these additions, the increase in the number of parameters remains modest. The parameters for $\theta_{\texttt{aug-N}}$ originate from the `MLP` defined in Eq. (5.25). In a configuration where the `MLP` has 3 layers with a feature dimension of d, the parameter count is $O(2d^2)$. The intervener ϕ, driven by the `transformer` layer in Eq. (5.27a), has its parameters confined to $O(3d^2 + 2d^2) = O(5d^2)$, owing to its query, key, value projection matrices and the feed-forward net (`FFN` from Eq. (5.24), typically a 3-layered `MLP`).

A step-by-step algorithm for FIG-N is in Algorithm 5.3. In test phase, the output of $\theta_{\texttt{pred}} \circ \texttt{Readout} \circ \phi(\mathbf{H}_{\texttt{ra}}||\mathbf{H}_{\texttt{env}})$ is evaluated.

5.2.5 Virtual Node-Level Variant: FIG-VN

In the previously introduced FIG-N, its augmenter decomposes the nodes into rationale nodes and environment nodes via a trainable node partitioner $\theta_{\texttt{aug-N}}$ so that the interaction

is conducted at the node level (Eqs. (5.27a)) whose dense attention matrix's complexity is quadratic in terms of the node numbers, infeasible for large graphs. This section extends this idea to extract the graph rationale at the virtual node level, which has a lower computation complexity compared to FIG-N and provides an intermediate intervention granularity between the node-level model (FIG-N) and the graph-level model (GREA [313]).

Virtual Node-Level Augmenter. Our idea is partly inspired by the speedup technique from Linformer [321], which reformulates both the attention matrix and node (token) embedding matrix to dimensions of $\mathbb{R}^{n\times r}$ and $\mathbb{R}^{r\times d}$, respectively. This reformulation ensures that their multiplication scales linearly with the number of nodes (tokens) n. Within this configuration, r, a pre-defined rank, is significantly smaller than n, and d represents the feature dimension. Drawing from the Linformer technique, we propose that the restructured token embedding matrix, with dimensions of $\mathbb{R}^{r\times d}$, can be interpreted as embeddings for r virtual nodes.

Building upon this insight, given node embeddings $\mathbf{H}$ from the encoder, the virtual node embeddings are:

$$\mathbf{H}_{\texttt{VN}} = \texttt{softmax}(\mathbf{W}_{\texttt{N-VN}})\mathbf{H}. \tag{5.33}$$

Here, the row-wise applied `softmax` function, along with
$\texttt{softmax}(\mathbf{W}_{\texttt{N-VN}}) \in \mathbb{R}^{r\times n}$, yields a trainable matrix assigning n nodes to r virtual nodes, where r acts as a tunable hyper-parameter. In experiments, we set $r = 8$. As all the virtual node embeddings are learned, a subset of the r virtual nodes can be designated as rationale virtual nodes, whose rationality is data-driven by the intervention procedure discussed in subsequent subsections. For brevity, the initial K virtual nodes are deemed as rationale virtual nodes, while the last $r - K$ nodes is considered the environment virtual node. Like the FIG-N, here K is a hyperparameter whose impact is studied in Section 5.2.6. Thus, rationale and environment embeddings are presented as:

$$\mathbf{H}_{\texttt{ra}} = \mathbf{H}_{\texttt{VN}}[:K,:] \in \mathbb{R}^{K\times d}, \tag{5.34a}$$

$$\mathbf{H}_{\texttt{env}} = \mathbf{H}_{\texttt{VN}}[K:,:] \in \mathbb{R}^{(r-K)\times d}. \tag{5.34b}$$

The parameter of $\theta_{\texttt{aug-VN}}$ is $\mathbf{W}_{\texttt{N-VN}}$.

Virtual Node-Level Intervener. This section discusses the design of a virtual node-level intervener, which parallels the framework presented in Section 5.2.4. The salient difference lies in that the intervention here functions on the virtual nodes rather than the given real nodes. Building upon our previous steps, we obtain the rationale virtual

node embeddings, $\mathbf{H}_{\texttt{ra}} \in \mathbb{R}^{K\times d}$, and the environment node embedding, $\mathbf{H}_{\texttt{env}} \in \mathbb{R}^{(r-K)\times d}$. Thanks to the property of the Transformer that it can process sets with variable size, the design of the virtual node-level intervener ϕ is similar to the node-level intervener as $\mathbf{H}_{\texttt{inter}}, \mathbf{P} = \texttt{transformer}(\mathbf{H}_{\texttt{ra}}||\mathbf{H}_{\texttt{env}})$ or short as $\mathbf{H}_{\texttt{inter}} = \phi(\mathbf{H}_{\texttt{ra}}||\mathbf{H}_{\texttt{env}})$ if the attention matrix $\mathbf{P}$ is not used. Notably, for FIG-VN, $\mathbf{P} \in \mathbb{R}^{r\times r}$ describes the interaction among the r virtual nodes.

FIG-VN Optimization Objective. The output of $\theta_{\texttt{pred}} \circ \texttt{readout} \circ \phi(\cdot)$ is used for minimizing the utility loss $\mathcal{L}_{\texttt{util}} = \mathcal{L}_{\texttt{util}}(\mathbf{H}_{\texttt{ra}}||\mathbf{H}_{\texttt{env}}) + \alpha\mathcal{L}_{\texttt{util}}(\mathbf{H}_{\texttt{ra}}||\tilde{\mathbf{H}}_{\texttt{env}})$, where $\mathcal{L}_{\texttt{util}}(\mathbf{H}_{\texttt{ra}}||\mathbf{H}_{\texttt{env}}) = \mathcal{L}_{\texttt{task}}(\theta_{\texttt{pred}} \circ \texttt{readout} \circ \phi(\mathbf{H}_{\texttt{ra}}||\mathbf{H}_{\texttt{env}}), \mathbf{y})$ and $\mathcal{L}_{\texttt{util}}(\mathbf{H}_{\texttt{ra}}||\tilde{\mathbf{H}}_{\texttt{env}})$ is defined similarly. For modeling the changing environment, $\tilde{\mathbf{H}}_{\texttt{env}}$ is the virtual node embeddings from other graphs in the mini-batch. Additionally, the previously proposed regularization term Eq, (5.29) can be extended to the virtual node-level variant: $\mathcal{L}_{\texttt{reg}}(\mathbf{H}_{\texttt{ra}}||\mathbf{H}_{\texttt{env}}) = \mathbf{s}^\top\mathbf{P}(\mathbf{1}-\mathbf{s}) + (\mathbf{1}-\mathbf{s})^\top\mathbf{P}\mathbf{s}$. The total regularization term, considering the changing environment $\tilde{\mathbf{H}}_{\texttt{env}}$, is $\mathcal{L}_{\texttt{reg}} = \mathcal{L}_{\texttt{reg}}(\mathbf{H}_{\texttt{ra}}||\mathbf{H}_{\texttt{env}}) + \mathcal{L}_{\texttt{reg}}(\mathbf{H}_{\texttt{ra}}||\tilde{\mathbf{H}}_{\texttt{env}})$. As the $\mathbf{P}$ depicts interactions among virtual nodes, we construct the rationale/environment indicator vector $\mathbf{s}$ analogously to Eq. (5.30).

Put everything together, and the optimization objective of FIG-VN is $\min_\theta \max_\phi \ \mathcal{L}_{\texttt{util}} + \beta\mathcal{L}_{\texttt{reg}}$, where $\theta = \{\theta_{\texttt{enc}}, \theta_{\texttt{aug-VN}}, \theta_{\texttt{pred}}\}$.

Complexity of FIG-VN. As we mentioned the encoder $\theta_{\texttt{enc}}$ and the predictor $\theta_{\texttt{pred}}$ are off-the-shelf. Thus, the extra modules introduced by the FIG-VN are the virtual node-level augmenter $\theta_{\texttt{aug-VN}}$ and the Transformer-based intervener ϕ. The parameters for $\theta_{\texttt{aug-VN}}$ originate from the matrix $\mathbf{W}_{\texttt{N-VN}}$, as defined in Eq. (5.33). The number of these parameters is in order $O(nr)$, where n denotes the number of nodes. For practical implementation purposes, n is pre-set; it is set to $10\times$ the average size of graphs from the dataset, and we truncate the input graphs if their size is larger than $10\times$ the average size. The intervener ϕ parameters originate from the `transformer` layer, outlined in Eq. (5.27a). The number of parameters here is $O(5d^2)$, owing to its query, key, value projection matrices, and the feed-forward net (Eq. (5.24), typically a 3-layered `MLP`).

A step-by-step algorithm for FIG-VN is in Algorithm 5.4. In test phase, the output of $\theta_{\texttt{pred}} \circ \texttt{Readout} \circ \phi(\mathbf{H}_{\texttt{ra}}||\mathbf{H}_{\texttt{env}})$ is evaluated.

5.2.6 Experiments

Setups. In this paper, we use 7 publicly-available real-world datasets: (1) graph classification datasets molhiv [268], moltox21 [268], molbace [268], molbbbp [268] and (2) graph

Algorithm 5.4: FIG-VN single training step for every training graph $\mathcal{G}$

Input : a labelled graph $(\mathcal{G}, y)$, a sampled graph $\tilde{\mathcal{G}}$ from the same batch as $\mathcal{G}$, $\theta = \{\theta_{\texttt{enc}}, \theta_{\texttt{aug-VN}}, \theta_{\texttt{pred}}\}$, ϕ;

Output: updated θ and ϕ;

compute the node embedding matrices $\mathbf{H} = \theta_{\texttt{enc}}(G)$ and $\tilde{\mathbf{H}} = \theta_{\texttt{enc}}(\tilde{G})$;

compute rationale and environment embeddings as $(\mathbf{H}_{\texttt{ra}}, \mathbf{H}_{\texttt{env}}) = \theta_{\texttt{aug-VN}}(\mathbf{H})$, $(\tilde{\mathbf{H}}_{\texttt{ra}}, \tilde{\mathbf{H}}_{\texttt{env}}) = \theta_{\texttt{aug-VN}}(\tilde{\mathbf{H}})$ via Eqs. (5.33), (5.34a), and (5.34b);

concatenate rationale-environment pairs $\mathbf{H}_{\texttt{ra}}||\mathbf{H}_{\texttt{env}}$ and $\mathbf{H}_{\texttt{ra}}||\tilde{\mathbf{H}}_{\texttt{env}}$;

compute $\mathcal{L}_{\texttt{util}}$ via Eq. (5.28);

compute $\mathcal{L}_{\texttt{reg}}$ via Eqs. (5.31) and (5.30);

update θ via gradient descent with $\frac{\partial(\mathcal{L}_{\texttt{util}}+\beta\mathcal{L}_{\texttt{reg}})}{\partial\theta}$;

update ϕ via gradient ascent with $\frac{\partial(\mathcal{L}_{\texttt{util}}+\beta\mathcal{L}_{\texttt{reg}})}{\partial\phi}$;

Table 5.6: Effectiveness comparison (mean±std) with baseline methods. (↓) denotes the lower the better and (↑) denotes the higher the better. Statistics in grey are reported in the original papers. The best is bold, and the second best is underlined. N/A means the method cannot work on regression tasks.

	Graph Regression			Graph Classification			
Dataset	ZINC	AQSOL	mollipo	molhiv	moltox21	molbace	molbbbp
Metric	MAE(↓)	MAE(↓)	RMSE(↓)	AUC(↑)	AUC(↑)	AUC(↑)	AUC(↑)
GIN	0.350±0.008	1.237±0.011	0.783±0.017	77.1±1.5	75.6±0.9	80.7±1.2	69.5±1.0
GAT	0.723±0.010	1.638±0.048	0.923±0.011	75.0±0.5	72.2±0.6	75.3±0.8	67.1±0.6
GATv2	0.729±0.015	1.722±0.022	0.943±0.021	72.2±0.5	73.6±0.2	76.8±1.6	65.7±0.7
GatedGCN	0.579±0.023	1.533±0.035	0.819±0.033	74.8±1.6	75.0±0.8	81.2±1.2	68.3±0.9
GT	0.226±0.014	1.319±0.026	0.882±0.020	73.5±0.4	75.0±0.6	77.1±2.3	65.0±1.1
GraphiT	0.202±0.011	1.162±0.005	0.846±0.023	74.6±1.0	71.8±1.3	73.4±3.6	64.6±0.5
SAN	0.139±0.006	1.199±0.218	0.816±0.112	77.9±0.2	71.3±0.8	79.0±3.1	63.8±0.9
SAT	0.094±0.008	1.236±0.023	0.835±0.008	78.8±0.6	75.6±0.7	83.6±2.1	69.6±1.3
Graphormer	0.122±0.006	1.265±0.025	0.911±0.015	79.3±0.4	77.3±0.8	79.3±3.0	67.7±0.9
GraphTrans	0.192±0.011	1.233±0.052	0.915±0.032	78.1±0.5	76.4±0.8	78.0±1.8	70.5±0.9
GPS	**0.070±0.004**	1.032±0.007	0.780±0.021	78.8±1.0	75.7±0.4	79.6±1.4	69.6±1.1
DIR	N/A	N/A	N/A	77.1±0.6	73.1±0.2	74.8±0.3	70.5±1.4
GREA	0.227±0.020	1.177±0.019	0.769±0.025	79.3±0.9	78.2±0.9	82.4±2.4	69.9±1.8
FIG-N	0.095±0.008	**0.990±0.012**	0.708±0.013	80.1±0.7	**78.8±0.5**	**85.3±2.0**	**73.8±0.7**
FIG-VN	0.086±0.012	1.011±0.009	**0.706±0.009**	**80.2±1.0**	78.2±0.6	84.5±1.3	73.1±0.8

regression datasets ZINC [328], AQSOL [328], and mollipo [268]. We strictly follow the metrics and dataset split recommended by the given benchmarks. To be concrete, the area under the ROC curve (AUC) is the metric for datasets molhiv, moltox21, molbace, molbbbp; root-mean-square deviation (RMSE) is the metric for dataset mollipo; mean absolute error (MAE) is the metric for datasets ZINC and AQSOL. The detailed statistics of the datasets

are given in Table 5.7. We report the average result with the standard deviation in 10 runs.

Table 5.7: Dataset statistics.

Name	# Graphs	# Nodes	# Edges	# Features	# Classes	Split	Metric
ZINC	12000	23.2	49.8	21 (node), 4 (edge)	N/A	10000/1000/1000	MAE
AQSOL	9833	17.6	35.8	65 (node), 5 (edge)	N/A	7836/998/999	MAE
mollipo	4200	27.0	59.0	9 (node), 3 (edge)	N/A	3360/420/420	RMSE
molhiv	41127	25.5	54.9	9 (node), 3 (edge)	2	32901/4113/4113	AUC
moltox21	7831	18.6	38.6	9 (node), 3 (edge)	2	6264/783/784	AUC
molbace	1513	34.4	73.7	9 (node), 3 (edge)	2	1210/151/152	AUC
molbbbp	2039	24.1	51.9	9 (node), 3 (edge)	2	1631/204/204	AUC
molmuv	93087	24.2	52.6	9 (node), 3 (edge)	2	74469/9309/9309	AUC

Our baseline methods include (1) 4 graph neural network models: GIN [329], GAT [330], GATv2 [331], and GatedGCN [332] (2) 7 graph Transformers: GT [333], GraphiT [334], SAN [335], SAT [326], Graphormer [336], GraphTrans [337], GPS [338], and (3) 2 graph rationale discovery methods: DIR [314] and GREA [313].

Effectiveness Study. The effectiveness comparison between the proposed FIG-N, FIG-VN, and baseline methods is provided in Table 5.6. To ensure a fair comparison, specific pre-trained models, such as the pre-trained Graphormer [336], are omitted. As DIR [314] is designed to conduct interventions on the label prediction vectors, it cannot be directly applied to graph regression tasks.

We have several observations. First, our proposed FIG-N and FIG-VN consistently outperform, or are at least on par with, all the baseline methods on the graph classification and regression datasets. Second, the virtual-node variant FIG-VN does not lead to significant performance degradation compared to the node-level variant FIG-N.

Efficiency Study. A detailed comparison regarding the number of parameters and the FLOPs (floating point operations) is presented in Table 5.8, where we list 4 typical 5-layered encoders (GIN, SAT, GraphTrans, and GPS), and our proposed node-/virtual node-level augmenter, intervener modules (i.e., $\{\theta_{\texttt{aug-N}}, \phi\}$ and $\{\theta_{\texttt{aug-VN}}, \phi\}$). The comparison shows that our proposed interveners are lightweight and incur only minor computational costs.

We also compare the model efficiency (training iterations/second) of FIG-N and FIG-VN in Table 5.9, working with different encoders (GIN and GPS). The batch size is set as 32. We note that the inclusion of our proposed augmenter and intervener, represented as $\{\theta_{\texttt{aug-N}}, \phi\}$ or $\{\theta_{\texttt{aug-VN}}, \phi\}$, introduces a slight reduction in training speed. That is because the proposed parametric augmenter and intervener increase the steps of the data pipeline, as presented in figure 5.4b, and enlarge the computational graph for auto-gradient tools, such

Table 5.8: Number of parameters and FLOPs comparisons between the proposed augmenter, intervener, and common graph encoders.

Model	# Parameters	FLOPs
GIN	$1,708,807$	$53,008,220$
SAT	$2,790,739$	$101,520,116$
GraphTrans	$2,793,307$	$111,548,906$
GPS	$3,236,239$	$133,229,235$
$\{\theta_{\texttt{aug-N}}, \phi\}$	$453,001$	$31,303,800$
$\{\theta_{\texttt{aug-VN}}, \phi\}$	$363,320$	$14,558,400$

Table 5.9: Wall-clock time (iterations/second) comparison of different encoder-intervener combinations. The larger, the faster. (↓) denotes the speed degradation compared with the vanilla encoder.

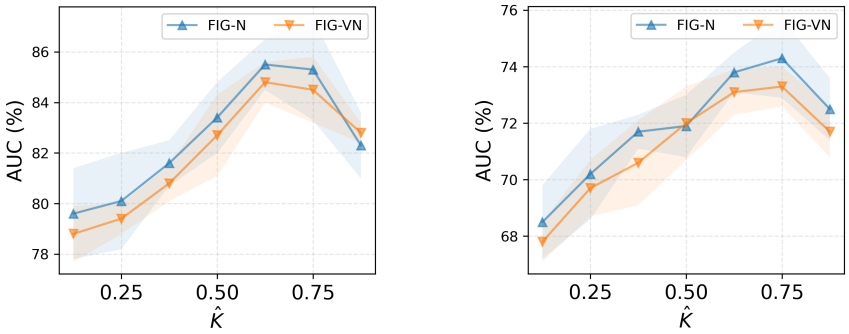
Encoder	Intervener	mollipo	molbace	molbbbp
GIN	None	29.38	25.62	27.11
	FIG-N	23.05(↓6.33)	21.23(↓4.39)	21.61(↓5.50)
	FIG-VN	23.35(↓6.03)	21.46(↓4.16)	22.32(↓4.79)
GPS	None	24.29	20.51	22.30
	FIG-N	19.57(↓4.72)	17.83(↓2.68)	18.67(↓3.63)
	FIG-VN	19.93(↓4.63)	18.16(↓2.35)	18.88(↓3.42)

as PyTorch. Fortunately, the parameter count of the parametric augmenter and intervener is low, ensuring that the overall training speed of the model is not dramatically affected.

Ablation Study. We conducted an ablation study on the proposed models, FIG-N and FIG-VN. We designed two ablated variants as baselines: (1) $\theta_{\texttt{enc}} \circ \theta_{\texttt{pred}}$ which is a pure composition of the encoder $\theta_{\texttt{enc}}$ and the predictor $\theta_{\texttt{pred}}$ without any rationale discovery module. Many of the existing graph classifiers are in this form, and here we select the GraphGPS [338], which is also the backbone of our FIG model. (2) FIG-N w/o reg and FIG-VN w/o reg which remove the regularization term (Eq. (5.31)) from the objective function. Our results in Table 5.10 highlight that (1) equipped with the proposed Transformer-based intervener, the model's performance improves across all the datasets; e.g., the AUC is improved from 79.6% to 84.3% (FIG-N) on the molbace dataset. (2) With the proposed regularization term, the model's performance can be improved further; e.g., the AUC of the FIG-N is improved from 72.3% to 73.8% on the molbbbp dataset.

Table 5.10: Ablation study (mean±std) of the proposed model FIG. (↓) denotes the lower the better and (↑) denotes the higher the better.

Dataset **Metric**	**mollipo** RMSE(↓)	**molbace** AUC(↑)	**molbbbp** AUC(↑)
$\theta_{\texttt{enc}} \circ \theta_{\texttt{pred}}$	0.780±0.021	79.6±1.4	70.5±0.9
FIG-N w/o reg	0.736±0.022	84.3±0.7	72.3±1.0
FIG-VN w/o reg	0.758±0.018	83.2±1.5	71.9±0.7
FIG-N w/ reg	0.708±0.013	85.3±2.0	73.8±0.7
FIG-VN w/ reg	0.706±0.009	84.5±1.3	73.1±0.8

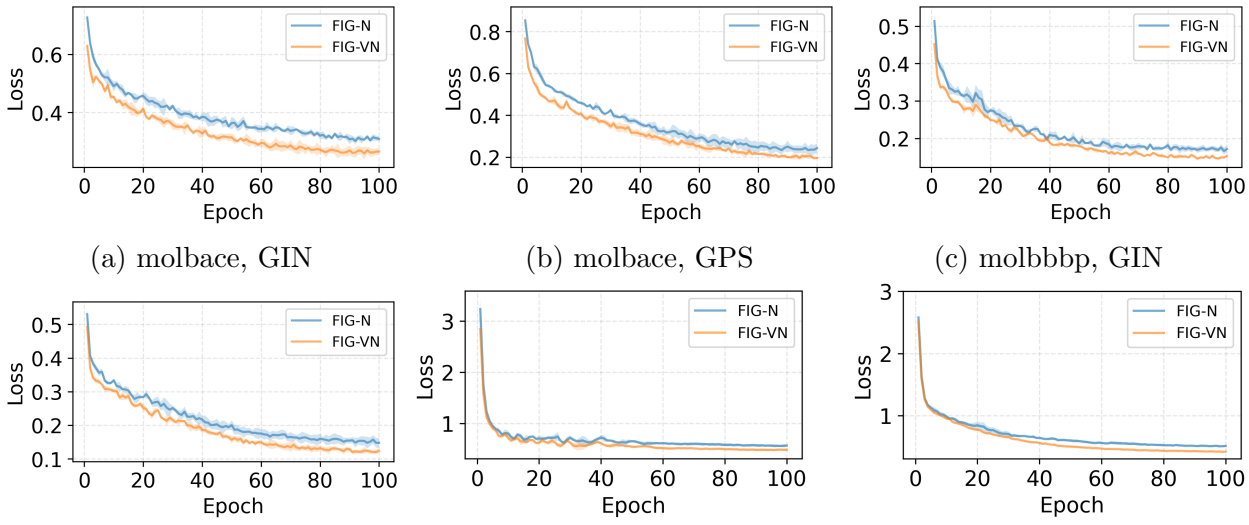


(a) molbace (b) molbbbp

Figure 5.5: Performance of FIG-N/VN with different $\hat{K}$.

Sensitivity Study. In this section, we carefully study the impact of hyperparameter K (from Eq. (5.26a), (5.26b), (5.34a), and (5.34b)), which determines the ratio of the rationale and environment subgraphs. In our implementation, we set $K = \texttt{round}(\hat{K} \times n)$ (for FIG-N) or $K = \texttt{round}(\hat{K} \times r)$ (for FIG-VN). We evaluate the model performance across varying $\hat{K}$ on the molbace and molbbbp datasets in Figure 5.5. We note that the model performance degrades if most nodes/virtual nodes are marked as the environment component. Similar performance degradation is observed if too many nodes/virtual nodes are marked as the rationale nodes (e.g., $\hat{K} = 0.875$). That is because for a large $\hat{K}$ (e.g., $\hat{K} = 1$), the model degenerates to a vanilla graph encoder, with less intervention involved. The best performance is observed when $\hat{K}$ is set as 0.75 or 0.675.

Training Convergence. We are concerned about the impact of the min-max objective on the training stability of FIG-N and FIG-VN. We monitor the training losses of both FIG-N and FIG-VN across three datasets (molbace, molbbbp, mollipo) using two encoders (GIN and GPS). The results, presented in Figure 5.6, demonstrate that the training remains stable even when θ (representing the encoder, augmenter, and predictor) and ϕ (representing the



(a) molbace, GIN (b) molbace, GPS (c) molbbbp, GIN

(d) molbbbp, GPS (e) mollipo, GIN (f) mollipo, GPS

Figure 5.6: Training loss of FIG-N/VN with different datasets and encoders.

intervener) engage in a min-max game.

Attention Visualization. In this section, we aim to evaluate the significance of the regularization term by visualizing the adjacency matrix $\mathbf{P}$, effectively the attention matrix, of the intervener ϕ in Figure 5.7. For clarity in visualization, we choose FIG-VN. Unlike FIG-N, which works on a variable number of nodes (from different graphs), FIG-VN maps nodes to a predetermined number of virtual nodes, simplifying the presentation. Specifically, we set the number of virtual nodes r to 16 with K at 10, designating 10 virtual nodes to rationales and the remainder as environments. All the visualization results are obtained from the molbace dataset. It is worth noting that the attention matrix $\mathbf{P}$ is normalized row-wise by `softmax`. From our observations, we highlight two primary insights:

- Interestingly, even in the absence of the regularization term, in Figure 5.7(a), interactions between rationales and environments appear significantly weaker than those within the rationales themselves. One potential explanation is the changing nature of the environment. In optimizing the utility loss $\mathcal{L}_{\texttt{util}} = \mathcal{L}_{\texttt{task}}(\mathbf{H}_{\texttt{ra}}||\mathbf{H}_{\texttt{env}})$ $+ \alpha\mathcal{L}_{\texttt{task}}(\mathbf{H}_{\texttt{ra}}||\tilde{\mathbf{H}}_{\texttt{env}})$, the ever-changing environment ($\tilde{\mathbf{H}}_{\texttt{env}}$) might lead the model to minimize interactions between rationales and environments so that the utility of the rationale can be ensured.

- The first observation supports our decision to introduce the regularization term, which aims to penalize rationale-environment interactions. When the proposed regularization term (Eq. (5.29)) is implemented, in Figure 5.7(b), there is a noticeable decline in

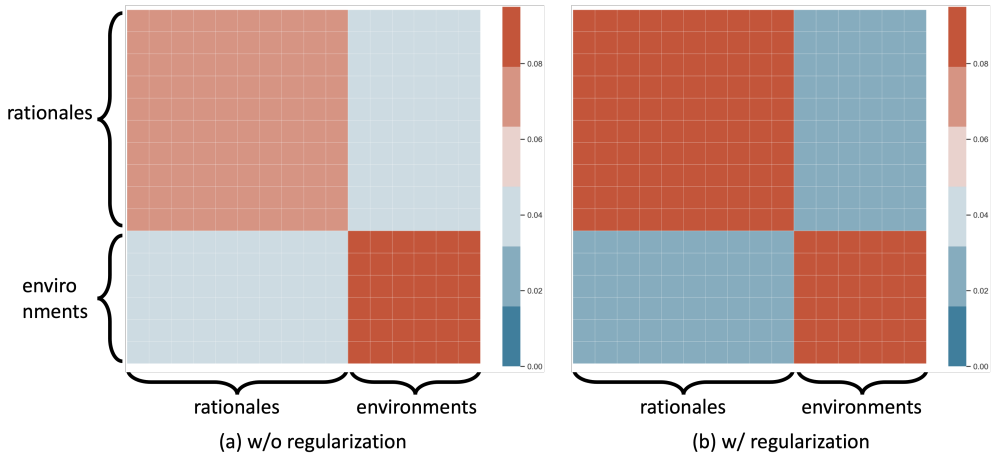


Figure 5.7: Heatmap of the adjacency matrix of the intervener ϕ. (a) without the regularization term Eq. (5.29) and (b) with the regularization term Eq. (5.29).

rationale-environment interactions (the off-diagonal blocks in Figure 5.7). As demonstrated in our earlier ablation study, this results in improved model performance.

Implementation of FIG-N/VN. The encoder is set as GPS [338] on ZINC, AQSOL, mollipo, molhiv, molbace, and set as GraphTrans [337] on moltox21 and molbbbp. We follow the typical design for the predictor module as a 3-layered MLP with ReLU activation in the intermediate layers. In our implementation, we set $\beta = \frac{2\times\hat{\beta}}{n\times(n-1)}$ (FIG-N) or $\beta = \frac{2\times\hat{\beta}}{r\times(r-1)}$ (FIG-VN). The α and $\hat{\beta}$ are searched between $[0.2, 2]$, step size 0.2. In our implementation, the K is set as $K = \texttt{round}(\hat{K} \times n)$ (for FIG-N) or $K = \texttt{round}(\hat{K} \times r)$ (for FIG-VN). r is searched between $\{8, 16, 32\}$ for FIG-VN. We have a detailed sensitivity study to explore the best selection of $\hat{K}$ in Section 5.2.6, which shows the best $\tilde{K}$ is around 0.75.

Implementation of Baseline Methods. We search the number of layers of GIN [329], GAT [330], GATv2 [331], GatedGCN [332], DIR [314], and GREA [313] between $\{2, 3, 5, 10\}$ and report the best performance, considering configurations both with and without a virtual node connecting to all the given nodes.

Regarding the Transformer-based baselines (GT [333], GraphiT [334], SAN [335], SAT [326], Graphormer [336], GraphTrans [337], GPS [338]), for the (absolute or relative) positional encoding, we adhere to the suggestions made in their original papers. We also searched the number of layers between $\{2, 3, 5, 10\}$.

Our GIN, GAT, GATv2, and GatedGCN implementations are from the PyTorch-geometric

package. Our implementations of GT [4], GraphiT [5], SAN [6], SAT [7], Graphormer [8], GraphTrans [9], GPS [10], DIR [11], GREA [12] are adapted from publicly available code.

5.2.7 Soft argtop-K Trick

In the main content, for the FIG-N model, the augmenter will generate the partition vector $\mathbf{m}$ which is then used to partition the node embedding matrix via selecting the top-K indices from $\mathbf{m}$:

$$\mathbf{m} = \texttt{sigmoid}(\texttt{MLP}(\mathbf{H}, \theta_{\texttt{aug-N}})) \in \mathbb{R}^n \tag{5.35}$$

$$\texttt{idx}_{\texttt{ra}} = \texttt{argtopK}(\mathbf{m}) \in \mathbb{N}_+^K \tag{5.36}$$

$$\mathbf{H}_{\texttt{ra}} = \mathbf{H}[\texttt{idx}_{\texttt{ra}}, :] \in \mathbb{R}^{K \times d} \tag{5.37}$$

$$\mathbf{H}_{\texttt{env}} = \mathbf{H}[\texttt{idx}_{\texttt{env}}, :] \in \mathbb{R}^{(n-K) \times d} \tag{5.38}$$

The hard `argtopK` breaks the differentiability of the model so that $\theta_{\texttt{aug-N}}$ has no gradient. Here, we present a soft argtop-K trick which is inspired by the soft top-K trick[13]. Overall, the key ideas are as follows,

1. The index vector $\texttt{idx}_{\texttt{ra}} \in \mathbb{N}_+^K$ can be viewed as a list of one-hot index encoding $\mathbf{idx} \in \{0,1\}^{K \times n}$ whose every row is a one-hot vector. If the i-th row's j-th element is 1, it means the j-th element in $\mathbf{m}$ is the i-th largest element in $\mathbf{m}$. Then, we can use the matrix multiplication to index the matrix $\mathbf{H}$:

$$\mathbf{H}[\texttt{idx}_{\texttt{ra}}, :] \Longleftrightarrow \mathbf{idx} \times \mathbf{H} \in \mathbb{R}^{K \times d} \tag{5.39}$$

2. We aim to find a soft and differentiable matrix to approximate $\mathbf{idx}$. The trick is to use the fact that every row of $\mathbf{idx}$ is one-hot, which can be approximated by the output of the softmax function. Hence, the implementation is to call the softmax K times repeatedly.

[4] `https://github.com/graphdeeplearning/graphtransformer`
[5] `https://github.com/inria-thoth/GraphiT`
[6] `https://github.com/DevinKreuzer/SAN`
[7] `https://github.com/BorgwardtLab/SAT`
[8] `https://github.com/microsoft/Graphormer`
[9] `https://github.com/ucbrise/graphtrans`
[10] `https://github.com/rampasek/GraphGPS`
[11] `https://github.com/Wuyxin/DIR-GNN`
[12] `https://github.com/liugangcode/GREA`
[13] `https://github.com/ZIB-IOL/merlin-arthur-classifiers/blob/main/soft-topk.py`

3. The parameterization trick[14] can be used: y_hard - y_soft.detach() + y_soft, whose main idea is to ensure (1) the forward process to use the hard indexing and (2) the backpropagation to update the soft indices.

[14] https://pytorch.org/docs/stable/generated/torch.nn.functional.gumbel_softmax.html

CHAPTER 6: CONCLUSION AND FUTURE WORK

6.1 CONCLUSION

This thesis advocates for a data-centric perspective in graph machine learning by introducing and advancing the concept of optimal graph learning. While the majority of prior work has focused on mining given graphs using increasingly complex models, this research shifts the emphasis toward optimizing the graph data itself to enhance the performance, efficiency, and expressiveness of downstream tasks.

We identified and addressed three core challenges inherent to this problem: formulating data optimization in a task-aware yet generalizable manner, managing the scalability issues posed by large graph volumes, and capturing essential structural patterns at multiple levels of granularity.

To tackle these challenges, we proposed a suite of solutions across three research directions: graph refinement, graph augmentation, and graph distillation. Our methods include data-driven techniques for noise reduction and class imbalance mitigation, structural augmentation approaches that enhance model generalizability and task compatibility, and distillation frameworks that reduce graph complexity while preserving task-relevant information.

Together, these contributions offer a coherent and principled foundation for optimizing graph data in machine learning workflows. They demonstrate that thoughtful manipulation of graph structure, beyond model design alone, can yield significant gains across a wide range of tasks. We hope this work inspires further research into data-centric methodologies for graph machine learning and contributes to the broader paradigm shift toward optimizing not just how we learn from graphs, but also what graphs we learn from.

6.2 FUTURE WORK

Data-centric graph machine learning remains an evolving field. Below, we outline several promising research directions.

Large Language Models for Graphs. The application of large language models (LLMs) across vision, language, and multimodal tasks represents a major step toward artificial general intelligence. Recently, initial efforts have explored the potential of LLMs in graph machine learningfor instance, leveraging LLMs to augment GNN training data [267], or evaluating LLMs ability to reason over graph structures [339]. A common conclusion in

these studies is that LLMs struggle to comprehend graph structures, largely due to their limited input context and the lack of inductive bias for graph topology.

This raises an important question: is the poor performance of LLMs on graph tasks primarily due to inadequate training data? Most LLMs are trained on general-purpose corpora, which do not typically include structured graph data. Consequently, they fail to perform even basic graph tasks such as identifying the largest connected component from an edge list. A potential avenue of research is to systematically generate labeled graph-centric tasks at scale with minimal human supervision. However, this remains non-trivial due to the combinatorial explosion in graph states with increasing node count, as well as the permutation invariance of graph representations.

Parallel-Sequential Hybrid Graph Generation. Current generative models can be broadly categorized into sequential (e.g., autoregressive language models) and parallel (e.g., diffusion or flow-based) approaches. Sequential models typically produce higher-quality outputs but are computationally expensive, while parallel models are more efficient but often sacrifice sample fidelity. Recent advances attempt to integrate both paradigms to achieve a balance of efficiency and quality, particularly in natural language [340] and image generation [341].

Graph data, by its nature, is hierarchical, making it a promising domain for parallel-sequential hybrid generation. For example, in social networks, communities can be modeled as subgraphs, with inter-community interactions forming a higher-level graph. Generation can then proceed in two stages: first, constructing the coarse-grained graph over clusters, followed by fine-grained generation within each cluster. However, this is a complex task. Unlike images, which can be represented as grid-like graphs with fixed dimensions, real-world graphs exhibit unbounded variability in both the number of clusters and the size of each subgraph. Furthermore, graph topology is fundamentally more intricate and less structured than other modalities, posing additional challenges.

REFERENCES

[1] R. Zafarani, M. A. Abbasi, and H. Liu, *Social media mining: an introduction*. Cambridge University Press, 2014.

[2] B. Jing, C. Park, and H. Tong, "Hdmi: High-order deep multiplex infomax," in *Proceedings of the Web Conference 2021*, 2021, pp. 2414–2424.

[3] D. Fu, Z. Xu, B. Li, H. Tong, and J. He, "A view-adversarial framework for multi-view network embedding," in *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*, M. d'Aquin, S. Dietze, C. Hauff, E. Curry, and P. Cudré-Mauroux, Eds. ACM, 2020. [Online]. Available: https://doi.org/10.1145/3340531.3412127 pp. 2025–2028.

[4] Q. Chen, N. Chen, W. Shuai, G. Wu, Z. Xu, H. Tong, and N. Cao, "Calliope-net: Automatic generation of graph data facts via annotated node-link diagrams," *IEEE Trans. Vis. Comput. Graph.*, vol. 30, no. 1, pp. 562–572, 2024. [Online]. Available: https://doi.org/10.1109/TVCG.2023.3326925

[5] H. Chen, Z. Xu, C. M. Yeh, V. Lai, Y. Zheng, M. Xu, and H. Tong, "Masked graph transformer for large-scale recommendation," in *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2024, Washington DC, USA, July 14-18, 2024*, G. H. Yang, H. Wang, S. Han, C. Hauff, G. Zuccon, and Y. Zhang, Eds. ACM, 2024. [Online]. Available: https://doi.org/10.1145/3626772.3657971 pp. 2502–2506.

[6] Z. Xu, S. Zhang, Y. Xia, L. Xiong, J. Xu, and H. Tong, "DESTINE: dense subgraph detection on multi-layered networks," in *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*, G. Demartini, G. Zuccon, J. S. Culpepper, Z. Huang, and H. Tong, Eds. ACM, 2021. [Online]. Available: https://doi.org/10.1145/3459637.3482083 pp. 3558–3562.

[7] Z. Xu, S. Zhang, Y. Xia, L. Xiong, and H. Tong, "Ranking on network of heterogeneous information networks," in *2020 IEEE International Conference on Big Data (IEEE BigData 2020), Atlanta, GA, USA, December 10-13, 2020*, X. Wu, C. Jermaine, L. Xiong, X. Hu, O. Kotevska, S. Lu, W. Xu, S. Aluru, C. Zhai, E. Al-Masri, Z. Chen, and J. Saltz, Eds. IEEE, 2020. [Online]. Available: https://doi.org/10.1109/BigData50022.2020.9378121 pp. 848–857.

[8] B. Alper, B. Bach, N. Henry Riche, T. Isenberg, and J.-D. Fekete, "Weighted graph comparison techniques for brain connectivity analysis," in *SIGCHI*, 2013, pp. 483–492.

[9] M. J. Keeling and K. T. Eames, "Networks and epidemic models," *Journal of the Royal Society Interface*, vol. 2, no. 4, pp. 295–307, 2005.

[10] D. Wang, J. Lin, P. Cui, Q. Jia, Z. Wang, Y. Fang, Q. Yu, J. Zhou, S. Yang, and Y. Qi, "A semi-supervised graph attentive network for financial fraud detection," in *ICDM*. IEEE, 2019, pp. 598–607.

[11] B. Jing, H. Tong, and Y. Zhu, "Network of tensor time series," in *Proceedings of the Web Conference 2021*, 2021, pp. 2425–2437.

[12] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

[13] G. Jeh and J. Widom, "Scaling personalized web search," in *WWW*, 2003, pp. 271–279.

[14] H. Tong, C. Faloutsos, and J.-Y. Pan, "Fast random walk with restart and its applications," in *ICDM*, 2006.

[15] T. H. Haveliwala, "Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search," *IEEE transactions on knowledge and data engineering*, vol. 15, no. 4, pp. 784–796, 2003.

[16] Z. Gyongyi, H. Garcia-Molina, and J. Pedersen, "Combating web spam with trustrank," in *VLDB*, 2004.

[17] R. Qiu, D. Wang, L. Ying, H. V. Poor, Y. Zhang, and H. Tong, "Reconstructing graph diffusion history from a single snapshot," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2023, Long Beach, CA, USA, August 6-10, 2023*, A. K. Singh, Y. Sun, L. Akoglu, D. Gunopulos, X. Yan, R. Kumar, F. Ozcan, and J. Ye, Eds. ACM, 2023. [Online]. Available: https://doi.org/10.1145/3580305.3599488 pp. 1978–1988.

[18] Y. Yan, Y. Hu, Q. Zhou, S. Wu, D. Wang, and H. Tong, "Topological anonymous walk embedding: A new structural node embedding approach," in *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management, CIKM 2024, Boise, ID, USA, October 21-25, 2024*, E. Serra and F. Spezzano, Eds. ACM, 2024. [Online]. Available: https://doi.org/10.1145/3627673.3679565 pp. 2796–2806.

[19] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 8, pp. 888–905, 2000.

[20] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=SJU4ayYgl

[21] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *ICLR*, 2018.

[22] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *ICML*, 2019.

[23] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," in *ICLR*, 2018.

[24] D. Fu, Z. Xu, H. Tong, and J. He, "Natural and artificial dynamics in gnns: A tutorial," in *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining, WSDM 2023, Singapore, 27 February 2023 - 3 March 2023*, T. Chua, H. W. Lauw, L. Si, E. Terzi, and P. Tsaparas, Eds. ACM, 2023. [Online]. Available: https://doi.org/10.1145/3539597.3572726 pp. 1252–1255.

[25] B. Zhao, B. Du, Z. Xu, L. Li, and H. Tong, "Learning optimal propagation for graph neural networks," *CoRR*, vol. abs/2205.02998, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2205.02998

[26] D. Fu and J. He, "SDG: A simplified and dynamic graph neural network," in *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021*, F. Diaz, C. Shah, T. Suel, P. Castells, R. Jones, and T. Sakai, Eds. ACM, 2021. [Online]. Available: https://doi.org/10.1145/3404835.3463059 pp. 2273–2277.

[27] L. Zheng, B. Jing, Z. Li, Z. Zeng, T. Wei, M. Ai, X. He, L. Liu, D. Fu, J. You, H. Tong, and J. He, "Pyg-ssl: A graph self-supervised learning toolkit," *CoRR*, vol. abs/2412.21151, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2412.21151

[28] Y. Yan, Y. Chen, H. Chen, X. Li, Z. Xu, Z. Zeng, Z. Liu, and H. Tong, "Thegcn: Temporal heterophilic graph convolutional network," *CoRR*, vol. abs/2412.16435, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2412.16435

[29] Y. Yan, Y. Chen, H. Chen, M. Xu, M. Das, H. Yang, and H. Tong, "From trainable negative depth to edge heterophily in graphs," in *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., 2023. [Online]. Available: http://papers.nips.cc/paper_files/paper/2023/hash/de2d52c5cf2bea853ef39bb2e1535dde-Abstract-Conference.html

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

[31] Z. Xu, B. Du, and H. Tong, "Graph sanitation with application to node classification," in *Proceedings of the ACM Web Conference 2022*, 2022, pp. 1136–1147.

[32] J. Zhu, Y. Yan, L. Zhao, M. Heimann, L. Akoglu, and D. Koutra, "Beyond homophily in graph neural networks: Current limitations and effective designs," *NeurIPS*, 2020.

[33] Z. Xu, K. Hassani, S. Zhang, H. Zeng, M. Yasunaga, L. Wang, D. Fu, N. Yao, B. Long, and H. Tong, "Language models are graph learners," *CoRR*, vol. abs/2410.02296, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2410.02296

[34] X. He, D. Fu, H. Tong, R. Maciejewski, and J. He, "Temporal heterogeneous graph generation with privacy, utility, and efficiency," in *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. [Online]. Available: https://openreview.net/forum?id=tj5xJInWty

[35] G. Liu, T. Zhao, J. Xu, T. Luo, and M. Jiang, "Graph rationalization with environment-based augmentations," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 1069–1078.

[36] D. Fu and J. He, "DPPIN: A biological repository of dynamic protein-protein interaction network data," in *IEEE International Conference on Big Data, Big Data 2022, Osaka, Japan, December 17-20, 2022*, S. Tsumoto, Y. Ohsawa, L. Chen, D. V. den Poel, X. Hu, Y. Motomura, T. Takagi, L. Wu, Y. Xie, A. Abe, and V. Raghavan, Eds. IEEE, 2022. [Online]. Available: https://doi.org/10.1109/BigData55660.2022.10020904 pp. 5269–5277.

[37] C. Zheng, B. Zong, W. Cheng, D. Song, J. Ni, W. Yu, H. Chen, and W. Wang, "Robust graph representation learning via neural sparsification," in *ICML*, 2020.

[38] D. Luo, W. Cheng, W. Yu, B. Zong, J. Ni, H. Chen, and X. Zhang, "Learning to drop: Robust graph neural network via topological denoising," in *WSDM*, 2021.

[39] Z. Xu, K. Ding, Y. Wang, H. Liu, and H. Tong, "Generalized few-shot node classification," in *IEEE International Conference on Data Mining, ICDM 2022, Orlando, FL, USA, November 28 - Dec. 1, 2022*, X. Zhu, S. Ranka, M. T. Thai, T. Washio, and X. Wu, Eds. IEEE, 2022. [Online]. Available: https://doi.org/10.1109/ICDM54844.2022.00071 pp. 608–617.

[40] J. Park, H. Shim, and E. Yang, "Graph transplant: Node saliency-guided graph mixup with local structure preservation," in *Proceedings of the First MiniCon Conference*, 2022.

[41] L. Wu, H. Lin, Z. Gao, C. Tan, S. Li et al., "Graphmixup: Improving class-imbalanced node classification on graphs by self-supervised context prediction," *arXiv preprint arXiv:2106.11133*, 2021.

[42] Z. Xu, Y. Chen, Q. Zhou, Y. Wu, M. Pan, H. Yang, and H. Tong, "Node classification beyond homophily: Towards a general solution," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023, pp. 2862–2873.

[43] Z. Xu, R. Qiu, Y. Chen, H. Chen, X. Fan, M. Pan, Z. Zeng, M. Das, and H. Tong, "Discrete-state continuous-time diffusion for graph generation," in *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, A. Globersons, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. M. Tomczak, and C. Zhang, Eds., 2024. [Online]. Available: http://papers.nips.cc/paper_files/paper/2024/hash/91813e5ddd9658b99be4c532e274b49c-Abstract-Conference.html

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

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

[46] Z. Xu, Y. Chen, M. Pan, H. Chen, M. Das, H. Yang, and H. Tong, "Kernel ridge regression-based graph dataset distillation," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023, pp. 2850–2861.

[47] Z. Xu, M. Pan, Y. Chen, H. Chen, Y. Yan, M. Das, and H. Tong, "Invariant graph transformer," *CoRR*, vol. abs/2312.07859, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2312.07859

[48] B. Hooi, H. A. Song, A. Beutel, N. Shah, K. Shin, and C. Faloutsos, "Fraudar: Bounding graph fraud in the face of camouflage," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 895–904.

[49] L. Chen, Y. Yao, F. Xu, M. Xu, and H. Tong, "Trading personalization for accuracy: Data debugging in collaborative filtering," *NeurIPS*, vol. 33, 2020.

[50] W. Bao, Z. Zeng, Z. Liu, H. Tong, and J. He, "Matcha: Mitigating graph structure shifts with test-time adaptation," in *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. [Online]. Available: https://openreview.net/forum?id=EpgoFFUM2q

[51] T. Zhao, Y. Liu, L. Neves, O. Woodford, M. Jiang, and N. Shah, "Data augmentation for graph neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 12, 2021, pp. 11 015–11 023.

[52] Y. Chen, L. Wu, and M. Zaki, "Iterative deep graph learning for graph neural networks: Better and robust node embeddings," *NeurIPS*, 2020.

[53] L. Yang, Z. Kang, X. Cao, D. Jin, B. Yang, and Y. Guo, "Topology optimization based graph convolutional network," in *IJCAI*, 2019.

[54] L. Franceschi, M. Niepert, M. Pontil, and X. He, "Learning discrete structures for graph neural networks," in *ICML*. PMLR, 2019, pp. 1972–1982.

[55] Y. Zhang, S. Pal, M. Coates, and D. Ustebay, "Bayesian graph convolutional neural networks for semi-supervised classification," in *AAAI*, 2019.

[56] R. Wang, S. Mou, X. Wang, W. Xiao, Q. Ju, C. Shi, and X. Xie, "Graph structure estimation neural networks," in *TheWebConf*, 2021.

[57] X. Liu, J. Ding, W. Jin, H. Xu, Y. Ma, Z. Liu, and J. Tang, "Graph neural networks with adaptive residual," *NeurIPS*, 2021.

[58] E. Rossi, H. Kenlay, M. I. Gorinova, B. P. Chamberlain, X. Dong, and M. Bronstein, "On the unreasonable effectiveness of feature propagation in learning on graphs with missing node features," *arXiv preprint arXiv:2111.12128*, 2021.

[59] H. Taguchi, X. Liu, and T. Murata, "Graph convolutional networks for graphs containing missing features," *FGCS*, 2021.

[60] X. Chen, H. Dai, Y. Li, X. Gao, and L. Song, "Learning to stop while learning to predict," in *International Conference on Machine Learning*. PMLR, 2020, pp. 1520–1530.

[61] D. Jin, C. Huo, C. Liang, and L. Yang, "Heterogeneous graph neural network via attribute completion," in *TheWebConf*, 2021.

[62] D. He, C. Liang, C. Huo, Z. Feng, D. Jin, L. Yang, and W. Zhang, "Analyzing heterogeneous networks with missing attributes by unsupervised contrastive learning," *TNNLS*, 2022.

[63] Z. Liu, R. Qiu, Z. Zeng, H. Yoo, D. Zhou, Z. Xu, Y. Zhu, K. Weldemariam, J. He, and H. Tong, "Class-imbalanced graph learning without class rebalancing," in *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. [Online]. Available: https://openreview.net/forum?id=pPnkpvBeZN

[64] D. Yarowsky, "Unsupervised word sense disambiguation rivaling supervised methods," in *ACL*, 1995.

[65] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *AAAI*, 2018.

[66] B. Hui, P. Zhu, and Q. Hu, "Collaborative graph convolutional networks: Unsupervised learning meets semi-supervised learning," in *AAAI*, 2020.

[67] K. Sun, Z. Lin, and Z. Zhu, "Multi-stage self-supervised learning for graph convolutional networks on graphs with few labeled nodes," in *AAAI*, 2020.

[68] E. Dai, C. Aggarwal, and S. Wang, "Nrgnn: Learning a label noise-resistant graph neural network on sparsely and noisily labeled graphs," in *KDD*, 2021.

[69] H. Dong, J. Chen, F. Feng, X. He, S. Bi, Z. Ding, and P. Cui, "On the equivalence of decoupled graph convolution network and label propagation," in *TheWebConf*, 2021.

[70] K. Ding, J. Wang, J. Caverlee, and H. Liu, "Meta propagation networks for graph few-shot semi-supervised learning," in *AAAI*, 2022.

[71] A. Blum and T. Mitchell, "Combining labeled and unlabeled data with co-training," in *COLT*, 1998.

[72] S. Li, W.-T. Li, and W. Wang, "Co-gcn for multi-view semi-supervised learning," in *AAAI*, 2020.

[73] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," in *ICLR*, 2018.

[74] Z. Zeng, R. Qiu, Z. Xu, Z. Liu, Y. Yan, T. Wei, L. Ying, J. He, and H. Tong, "Graph mixup on approximate gromov-wasserstein geodesics," in *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. [Online]. Available: https://openreview.net/forum?id=PKdege0U6Z

[75] Q. Zhou, Y. Chen, Z. Xu, Y. Wu, M. Pan, M. Das, H. Yang, and H. Tong, "Graph anomaly detection with adaptive node mixup," in *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management, CIKM 2024, Boise, ID, USA, October 21-25, 2024*, E. Serra and F. Spezzano, Eds. ACM, 2024. [Online]. Available: https://doi.org/10.1145/3627673.3679577 pp. 3494–3504.

[76] V. Verma, A. Lamb, C. Beckham, A. Najafi, I. Mitliagkas, D. Lopez-Paz, and Y. Bengio, "Manifold mixup: Better representations by interpolating hidden states," in *ICML*, 2019.

[77] V. Verma, M. Qu, K. Kawaguchi, A. Lamb, Y. Bengio, J. Kannala, and J. Tang, "Graphmix: Improved training of gnns for semi-supervised learning," in *AAAI*, 2021.

[78] Y. Wang, W. Wang, Y. Liang, Y. Cai, and B. Hooi, "Mixup for node and graph classification," in *TheWebConf*, 2021.

[79] H. Guo and Y. Mao, "ifmixup: Towards intrusion-free graph mixup for graph classification," *arXiv preprint arXiv:2110.09344*, 2021.

[80] X. Han, Z. Jiang, N. Liu, and X. Hu, "G-mixup: Graph data augmentation for graph classification," in *ICML*, 2022.

[81] J. Chen, Y. Saad, and Z. Zhang, "Graph coarsening: from scientific computing to machine learning," *SeMA Journal*, vol. 79, no. 1, pp. 187–223, 2022.

[82] W. Jin, Y. Ma, X. Liu, X. Tang, S. Wang, and J. Tang, "Graph structure learning for robust graph neural networks," in *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, 2020, pp. 66–74.

[83] C. Cai, D. Wang, and Y. Wang, "Graph coarsening with neural networks," in *International Conference on Learning Representations*, 2021.

[84] R. Yu, S. Liu, and X. Wang, "Dataset distillation: A comprehensive review," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[85] T.-W. Li, R. Qiu, and H. Tong, "Model-free graph data selection under distribution shift," *arXiv preprint arXiv:2505.17293*, 2025.

[86] Z. Zeng, X. Liu, M. Hang, X. Liu, Q. Zhou, C. Yang, Y. Liu, Y. Ruan, L. Chen, Y. Chen, Y. Hao, J. Xu, J. Nie, X. Liu, B. Zhang, W. Wen, S. Yuan, K. Wang, W. Chen, Y. Han, H. Li, C. Yang, B. Long, P. S. Yu, H. Tong, and J. Yang, "Interformer: Towards effective heterogeneous interaction learning for click-through rate prediction," *CoRR*, vol. abs/2411.09852, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2411.09852

[87] X. Lin, Z. Liu, D. Fu, R. Qiu, and H. Tong, "Backtime: Backdoor attacks on multivariate time series forecasting," in *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, A. Globersons, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. M. Tomczak, and C. Zhang, Eds., 2024. [Online]. Available: http://papers.nips.cc/paper_files/paper/2024/hash/ed3cd2520148b577039adfade82a5566-Abstract-Conference.html

[88] Q. Zhou, L. Li, N. Cao, L. Ying, and H. Tong, "Adversarial attacks on multi-network mining: Problem definition and fast solutions," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 1, pp. 96–107, 2023. [Online]. Available: https://doi.org/10.1109/TKDE.2021.3078634

[89] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *Journal of the American society for information science and technology*, vol. 58, no. 7, pp. 1019–1031, 2007.

[90] M. Huisman, "Imputation of missing network data: Some simple procedures," *Journal of Social Structure*, vol. 10, no. 1, pp. 1–29, 2009.

[91] D. Wang, Y. Yan, R. Qiu, Y. Zhu, K. Guan, A. Margenot, and H. Tong, "Networked time series imputation via position-aware graph enhanced variational autoencoders," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2023, Long Beach, CA, USA, August 6-10, 2023*, A. K. Singh, Y. Sun, L. Akoglu, D. Gunopulos, X. Yan, R. Kumar, F. Ozcan, and J. Ye, Eds. ACM, 2023. [Online]. Available: https://doi.org/10.1145/3580305.3599444 pp. 2256–2268.

[92] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *NIPS*, 2013, pp. 1–9.

[93] Z. Wang, J. Zhang, J. Feng, and Z. Chen, "Knowledge graph embedding by translating on hyperplanes," in *AAAI*, vol. 28, no. 1, 2014.

[94] L. Liu, B. Du, Y. R. Fung, H. Ji, J. Xu, and H. Tong, "Kompare: A knowledge graph comparative reasoning system," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 3308–3318.

[95] L. Liu, B. Du, H. Ji, C. Zhai, and H. Tong, "Neural-answering logical queries on knowledge graphs," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 1087–1097.

[96] Y. Yan, L. Liu, Y. Ban, B. Jing, and H. Tong, "Dynamic knowledge graph alignment," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 5, 2021, pp. 4564–4572.

[97] H. Jiang, L. Cui, Z. Xu, D. Yang, J. Chen, C. Li, J. Liu, J. Liang, C. Wang, Y. Xiao, and W. Wang, "Relation extraction using supervision from topic knowledge of relation labels," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, S. Kraus, Ed. ijcai.org, 2019. [Online]. Available: https://doi.org/10.24963/ijcai.2019/698 pp. 5024–5030.

[98] C. Chen, R. Peng, L. Ying, and H. Tong, "Network connectivity optimization: Fundamental limits and effective algorithms," in *SIGKDD*, 2018, pp. 1167–1176.

[99] C. Chen, H. Tong, B. A. Prakash, C. E. Tsourakakis, T. Eliassi-Rad, C. Faloutsos, and D. H. Chau, "Node immunization on large graphs: Theory and algorithms," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 1, pp. 113–126, 2015.

[100] N. Entezari, S. A. Al-Sayouri, A. Darvishzadeh, and E. E. Papalexakis, "All you need is low (rank) defending against adversarial attacks on graphs," in *Proceedings of the 13th International Conference on Web Search and Data Mining*, 2020, pp. 169–177.

[101] H. Wu, C. Wang, Y. Tyshetskiy, A. Docherty, K. Lu, and L. Zhu, "Adversarial examples for graph data: deep insights into attack and defense," in *IJCAI*. AAAI Press, 2019, pp. 4816–4823.

[102] W. Jin, Y. Ma, X. Liu, X. Tang, S. Wang, and J. Tang, "Graph structure learning for robust graph neural networks," in *SIGKDD*. ACM, 2020, pp. 66–74.

[103] D. Zhu, Z. Zhang, P. Cui, and W. Zhu, "Robust graph convolutional networks against adversarial attacks," in *SIGKDD*, 2019, pp. 1399–1407.

[104] D. Zügner and S. Günnemann, "Adversarial attacks on graph neural networks via meta learning," in *ICLR*, 2019.

[105] J. Kang and H. Tong, "N2n: Network derivative mining," in *CIKM*, 2019, pp. 861–870.

[106] J. Kang, J. He, R. Maciejewski, and H. Tong, "Inform: Individual fairness on graph mining," in *SIGKDD*, 2020, pp. 379–389.

[107] L. Backstrom and J. Leskovec, "Supervised random walks: predicting and recommending links in social networks," in *WSDM*, 2011, pp. 635–644.

[108] L. Li, Y. Yao, J. Tang, W. Fan, and H. Tong, "Quint: on query-specific optimal networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 985–994.

[109] X. Wang and I. Davidson, "Flexible constrained spectral clustering," in *SIGKDD*, 2010, pp. 563–572.

[110] B. Du, C. Yuan, R. Barton, T. Neiman, and H. Tong, "Hypergraph pre-training with graph neural networks," *arXiv preprint arXiv:2105.10862*, 2021.

[111] Z. Li, D. Fu, and J. He, "Everything evolves in personalized pagerank," in *Proceedings of the ACM Web Conference 2023, WWW 2023, Austin, TX, USA, 30 April 2023 - 4 May 2023*, Y. Ding, J. Tang, J. F. Sequeda, L. Aroyo, C. Castillo, and G. Houben, Eds. ACM, 2023. [Online]. Available: https://doi.org/10.1145/3543507.3583474 pp. 3342–3352.

[112] L. Yan, R. Dodier, M. C. Mozer, and R. Wolniewicz, "Optimizing classifier performance via an approximation to the wilcoxon-mann-whitney statistic," in *ICML*, 2003, pp. 848–855.

[113] K. Wagstaff and C. Cardie, "Clustering with instance-level constraints," *AAAI/IAAI*, vol. 1097, pp. 577–584, 2000.

[114] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," in *SIGKDD*, 2018, pp. 2847–2856.

[115] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: a survey," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 5595–5637, 2017.

[116] A. Shaban, C.-A. Cheng, N. Hatch, and B. Boots, "Truncated back-propagation for bilevel optimization," in *AISTATS*. PMLR, 2019, pp. 1723–1732.

[117] A. Nichol, J. Achiam, and J. Schulman, "On first-order meta-learning algorithms," *arXiv preprint arXiv:1803.02999*, 2018.

[118] L. Van Der Maaten, "Accelerating t-sne using tree-based algorithms," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3221–3245, 2014.

[119] Y. Rong, W. Huang, T. Xu, and J. Huang, "Dropedge: Towards deep graph convolutional networks on node classification," in *ICLR*, 2019.

[120] C. C. Aggarwal and N. Li, "On node classification in dynamic content-based networks," in *SDM*, 2011.

[121] E. Chien, J. Peng, P. Li, and O. Milenkovic, "Adaptive universal generalized pagerank graph neural network," in *ICLR*, 2021.

[122] J. Tang, C. Aggarwal, and H. Liu, "Node classification in signed social networks," in *SDM*, 2016.

[123] S. Bhagat, G. Cormode, and S. Muthukrishnan, "Node classification in social networks," in *Social network data analytics*. Springer, 2011, pp. 115–148.

[124] K. Ding, J. Wang, J. Li, K. Shu, C. Liu, and H. Liu, "Graph prototypical networks for few-shot learning on attributed networks," in *CIKM*, 2020.

[125] F. Zhou, C. Cao, K. Zhang, G. Trajcevski, T. Zhong, and J. Geng, "Meta-gnn: On few-shot node classification in graph meta-learning," in *CIKM*, 2019.

[126] Z. Liu, Y. Fang, C. Liu, and S. C. Hoi, "Relative and absolute location embedding for few-shot node classification on graph," in *AAAI*, 2021.

[127] K. Huang and M. Zitnik, "Graph meta learning via local subgraphs," *NeurIPS*, 2020.

[128] K. Ding, Q. Zhou, H. Tong, and H. Liu, "Few-shot network anomaly detection via cross-network meta-learning," in *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, J. Leskovec, M. Grobelnik, M. Najork, J. Tang, and L. Zia, Eds. ACM / IW3C2, 2021. [Online]. Available: https://doi.org/10.1145/3442381.3449922 pp. 2448–2456.

[129] S. Thrun, "Lifelong learning algorithms," in *Learning to learn*. Springer, 1998, pp. 181–209.

[130] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra et al., "Matching networks for one shot learning," in *NIPS*, 2016.

[131] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," in *NIPS*, 2017.

[132] N. Wang, M. Luo, K. Ding, L. Zhang, J. Li, and Q. Zheng, "Graph few-shot learning with attribute matching," in *CIKM*, 2020.

[133] H.-J. Ye, H. Hu, and D.-C. Zhan, "Learning adaptive classifiers synthesis for generalized few-shot learning," *International Journal of Computer Vision*, pp. 1–24, 2021.

[134] Z. Liu, Z. Wei, E. Yu, Q. Huang, K. Guo, B. Yu, Z. Cai, H. Ye, W. Cao, J. Bian, P. Wei, J. Jiang, and Y. Chang, "IMBENS: ensemble class-imbalanced learning in python," *CoRR*, vol. abs/2111.12776, 2021. [Online]. Available: https://arxiv.org/abs/2111.12776

[135] H. Wang, B. Jing, K. Ding, Y. Zhu, W. Cheng, S. Zhang, Y. Fan, L. Zhang, and D. Zhou, "Mastering long-tail complexity on graphs: Characterization, learning, and generalization," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2024, Barcelona, Spain, August 25-29, 2024*, R. Baeza-Yates and F. Bonchi, Eds. ACM, 2024. [Online]. Available: https://doi.org/10.1145/3637528.3671880 pp. 3045–3056.

[136] J. McAuley, R. Pandey, and J. Leskovec, "Inferring networks of substitutable and complementary products," in *SIGKDD*, 2015.

[137] A. Bojchevski and S. Günnemann, "Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking," in *ICLR*, 2018.

[138] W.-L. Chao, S. Changpinyo, B. Gong, and F. Sha, "An empirical study and analysis of generalized zero-shot learning for object recognition in the wild," in *ECCV*, 2016.

[139] R. Felix, I. Reid, G. Carneiro et al., "Multi-modal cycle-consistent generalized zero-shot learning," in *ECCV*, 2018.

[140] Y. Yu, Z. Ji, J. Han, and Z. Zhang, "Episode-based prototype generating network for zero-shot learning," in *CVPR*, 2020.

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

[142] H. Yao, C. Zhang, Y. Wei, M. Jiang, S. Wang, J. Huang, N. Chawla, and Z. Li, "Graph few-shot learning via knowledge transfer," in *AAAI*, 2020.

[143] K. Ding, Q. Zhou, H. Tong, and H. Liu, "Few-shot network anomaly detection via cross-network meta-learning," in *TheWebConf*, 2021.

[144] A. Kendall and Y. Gal, "What uncertainties do we need in bayesian deep learning for computer vision?" in *NIPS*, 2017.

[145] A. Alaa and M. Van Der Schaar, "Discriminative jackknife: Quantifying uncertainty in deep learning via higher-order influence functions," in *ICML*, 2020.

[146] Y. Gal and Z. Ghahramani, "Bayesian convolutional neural networks with bernoulli approximate variational inference," *arXiv preprint arXiv:1506.02158*, 2015.

[147] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *JMLR*, 2014.

[148] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, 1989.

[149] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *ICML*, 2017.

[150] L. Franceschi, M. Donini, P. Frasconi, and M. Pontil, "Forward and reverse gradient-based hyperparameter optimization," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1165–1173.

[151] L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil, "Bilevel programming for hyperparameter optimization and meta-learning," in *ICML*. PMLR, 2018.

[152] E. Triantafillou, T. Zhu, V. Dumoulin, P. Lamblin, U. Evci, K. Xu, R. Goroshin, C. Gelada, K. Swersky, P. Manzagol, and H. Larochelle, "Meta-dataset: A dataset of datasets for learning to learn from few examples," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. [Online]. Available: https://openreview.net/forum?id=rkgAGAVKPr

[153] H. Lee, H. Lee, D. Na, S. Kim, M. Park, E. Yang, and S. J. Hwang, "Learning to balance: Bayesian meta-learning for imbalanced and out-of-distribution tasks," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. [Online]. Available: https://openreview.net/forum?id=rkeZIJBYvr

[154] J. Guan, J. Liu, J. Sun, P. Feng, T. Shuai, and W. Wang, "Meta metric learning for highly imbalanced aerial scene classification," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 4047–4051.

[155] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su, "Arnetminer: extraction and mining of academic social networks," in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2008, pp. 990–998.

[156] M. Buda, A. Maki, and M. A. Mazurowski, "A systematic study of the class imbalance problem in convolutional neural networks," *Neural networks*, vol. 106, pp. 249–259, 2018.

[157] T. Zhao, X. Zhang, and S. Wang, "Graphsmote: Imbalanced node classification on graphs with graph neural networks," in *Proceedings of the 14th ACM international conference on web search and data mining*, 2021, pp. 833–841.

[158] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

[159] D. Zhou, L. Zheng, D. Fu, J. Han, and J. He, "Mentorgnn: Deriving curriculum for pre-training gnns," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management, Atlanta, GA, USA, October 17-21, 2022*, M. A. Hasan and L. Xiong, Eds. ACM, 2022. [Online]. Available: https://doi.org/10.1145/3511808.3557393 pp. 2721–2731.

[160] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *International Conference on Learning Representations*, 2018.

[161] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," *NeurIPS*, 2018.

[162] Y. Ban, J. Zou, Z. Li, Y. Qi, D. Fu, J. Kang, H. Tong, and J. He, "Pagerank bandits for link prediction," in *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, A. Globersons, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. M. Tomczak, and C. Zhang, Eds., 2024. [Online]. Available: http://papers.nips.cc/paper_files/paper/2024/hash/25ead0efeed514aec00109301d93bbbb-Abstract-Conference.html

[163] F. M. Bianchi, D. Grattarola, and C. Alippi, "Spectral clustering with graph neural networks for graph pooling," in *ICML*, 2020.

[164] Z. Yang, W. Cohen, and R. Salakhudinov, "Revisiting semi-supervised learning with graph embeddings," in *ICML*, 2016.

[165] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in neural information processing systems*, 2017, pp. 1024–1034.

[166] E. Chien, J. Peng, P. Li, and O. Milenkovic, "Adaptive universal generalized pagerank graph neural network," in *International Conference on Learning Representations*, 2021.

[167] D. Bo, X. Wang, C. Shi, and H. Shen, "Beyond low-frequency information in graph convolutional networks," in *AAAI*, 2021.

[168] Y. Zhu, W. Xu, J. Zhang, Q. Liu, S. Wu, and L. Wang, "Deep graph structure learning for robust representations: A survey," *arXiv preprint arXiv:2103.03036*, 2021.

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

[170] M. Balcilar, G. Renton, P. Héroux, B. Gaüzère, S. Adam, and P. Honeine, "Analyzing the expressive power of graph neural networks in a spectral perspective," in *ICLR*, 2021.

[171] X. Zheng, Y. Liu, S. Pan, M. Zhang, D. Jin, and P. S. Yu, "Graph neural networks for graphs with heterophily: A survey," *arXiv preprint arXiv:2202.07082*, 2022.

[172] H. Pei, B. Wei, K. C.-C. Chang, Y. Lei, and B. Yang, "Geom-gcn: Geometric graph convolutional networks," in *ICLR*, 2019.

[173] H. Xu, Y. Yan, D. Wang, Z. Xu, Z. Zeng, T. F. Abdelzaher, J. Han, and H. Tong, "SLOG: an inductive spectral graph neural network beyond polynomial filter," in *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. [Online]. Available: https://openreview.net/forum?id=0SrNCSklZx

[174] L. Chen, Z. Chen, and J. Bruna, "On graph neural networks versus graph-augmented mlps," in *ICLR*, 2021.

[175] M. Newman, *Networks*. Oxford university press, 2018.

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

[177] T. Wang, D. Jin, R. Wang, D. He, and Y. Huang, "Powerful graph convolutional networks with adaptive propagation mechanism for homophily and heterophily," in *AAAI*, 2022.

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

[179] B. Rozemberczki, C. Allen, and R. Sarkar, "Multi-scale attributed node embedding," *Journal of Complex Networks*, vol. 9, no. 2, p. cnab014, 2021.

[180] D. Lim, F. Hohne, X. Li, S. L. Huang, V. Gupta, O. Bhalerao, and S. N. Lim, "Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods," *NeurIPS*, 2021.

[181] S. Luan, C. Hua, Q. Lu, J. Zhu, M. Zhao, S. Zhang, X.-W. Chang, and D. Precup, "Is heterophily a real nightmare for graph neural networks to do node classification?" *arXiv preprint arXiv:2109.05641*, 2021.

[182] M. He, Z. Wei, H. Xu et al., "Bernnet: Learning arbitrary graph spectral filters via bernstein approximation," *NeurIPS*, 2021.

[183] X. Li, R. Zhu, Y. Cheng, C. Shan, S. Luo, D. Li, and W. Qian, "Finding global homophily in graph neural networks when meeting heterophily," *arXiv preprint arXiv:2205.07308*, 2022.

[184] K. Madhawa, K. Ishiguro, K. Nakago, and M. Abe, "Graphnvp: An invertible flow model for generating molecular graphs," *CoRR*, vol. abs/1905.11600, 2019.

[185] Y. Kwon, D. Lee, Y. Choi, K. Shin, and S. Kang, "Compressed graph representation for scalable molecular graph generation," *J. Cheminformatics*, vol. 12, no. 1, p. 58, 2020.

[186] C. Niu, Y. Song, J. Song, S. Zhao, A. Grover, and S. Ermon, "Permutation invariant graph generation via score-based generative modeling," in *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]*, ser. Proceedings of Machine Learning Research, S. Chiappa and R. Calandra, Eds., vol. 108. PMLR, 2020, pp. 4474–4484.

[187] K. Martinkus, A. Loukas, N. Perraudin, and R. Wattenhofer, "SPECTRE: spectral conditioning helps to overcome the expressivity limits of one-shot graph generators," in *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu, and S. Sabato, Eds., vol. 162. PMLR, 2022, pp. 15 159–15 179.

[188] C. Vignac and P. Frossard, "Top-n: Equivariant set and graph generation without exchangeability," in *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

[189] J. Jo, S. Lee, and S. J. Hwang, "Score-based generative modeling of graphs via the system of stochastic differential equations," in *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu, and S. Sabato, Eds., vol. 162. PMLR, 2022, pp. 10 362–10 383.

[190] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec, "Graphrnn: Generating realistic graphs with deep auto-regressive models," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, ser. Proceedings of Machine Learning Research, J. G. Dy and A. Krause, Eds., vol. 80. PMLR, 2018, pp. 5694–5703.

[191] W. Jin, R. Barzilay, and T. S. Jaakkola, "Junction tree variational autoencoder for molecular graph generation," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, ser. Proceedings of Machine Learning Research, J. G. Dy and A. Krause, Eds., vol. 80. PMLR, 2018, pp. 2328–2337.

[192] R. Liao, Y. Li, Y. Song, S. Wang, W. L. Hamilton, D. Duvenaud, R. Urtasun, and R. S. Zemel, "Efficient graph generation with graph recurrent attention networks," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 4257–4267.

[193] R. Mercado, T. Rastemo, E. Lindelöf, G. Klambauer, O. Engkvist, H. Chen, and E. J. Bjerrum, "Graph networks for molecular design," *Mach. Learn. Sci. Technol.*, vol. 2, no. 2, p. 25023, 2021.

[194] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, ser. JMLR Workshop and Conference Proceedings, F. R. Bach and D. M. Blei, Eds., vol. 37. JMLR.org, 2015, pp. 2256–2265.

[195] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020.

[196] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, "Score-based generative modeling through stochastic differential equations," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[197] C. Vignac, I. Krawczuk, A. Siraudin, B. Wang, V. Cevher, and P. Frossard, "Digress: Discrete denoising diffusion for graph generation," in *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

[198] D. T. Gillespie, "Approximate accelerated stochastic simulation of chemically reacting systems," *The Journal of chemical physics*, vol. 115, no. 4, pp. 1716–1733, 2001.

[199] A. Campbell, J. Benton, V. D. Bortoli, T. Rainforth, G. Deligiannidis, and A. Doucet, "A continuous time framework for discrete denoising models," in *NeurIPS*, 2022.

[200] H. Sun, L. Yu, B. Dai, D. Schuurmans, and H. Dai, "Score-based continuous-time discrete diffusion models," in *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

[201] E. Min, R. Chen, Y. Bian, T. Xu, K. Zhao, W. Huang, P. Zhao, J. Huang, S. Ananiadou, and Y. Rong, "Transformer for graphs: An overview from architecture perspective," *CoRR*, vol. abs/2202.08455, 2022. [Online]. Available: https://arxiv.org/abs/2202.08455

[202] S. I. Resnick, *Adventures in stochastic processes*. Springer Science & Business Media, 1992.

[203] D. T. Gillespie, "A general method for numerically simulating the stochastic time evolution of coupled chemical reactions," *Journal of computational physics*, vol. 22, no. 4, pp. 403–434, 1976.

[204] D. T. Gillespie, "Exact stochastic simulation of coupled chemical reactions," *The journal of physical chemistry*, vol. 81, no. 25, pp. 2340–2361, 1977.

[205] D. F. Anderson, "A modified next reaction method for simulating chemical systems with time dependent propensities and delays," *The Journal of chemical physics*, vol. 127, no. 21, 2007.

[206] J. Austin, D. D. Johnson, J. Ho, D. Tarlow, and R. van den Berg, "Structured denoising diffusion models in discrete state-spaces," in *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021, pp. 17 981–17 993.

[207] D. Brook, "On the distinction between the conditional probability and the joint probability approaches in the specification of nearest-neighbour systems," *Biometrika*, vol. 51, no. 3/4, pp. 481–483, 1964.

[208] S. Lyu, "Interpretation and generalization of score matching," in *UAI 2009, Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, Montreal, QC, Canada, June 18-21, 2009*, J. A. Bilmes and A. Y. Ng, Eds. AUAI Press, 2009, pp. 359–366.

[209] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 2017, pp. 1263–1272.

[210] J. Kim, D. Nguyen, S. Min, S. Cho, M. Lee, H. Lee, and S. Hong, "Pure transformers are powerful graph learners," in *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., 2022.

[211] C. Cai, T. S. Hy, R. Yu, and Y. Wang, "On the connection between MPNN and graph transformer," in *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, ser. Proceedings of Machine Learning Research, A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, Eds., vol. 202. PMLR, 2023, pp. 3408–3430.

[212] Z. Chen, L. Chen, S. Villar, and J. Bruna, "Can graph neural networks count substructures?" in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020.

[213] G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Velickovic, "Principal neighbourhood aggregation for graph nets," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020.

[214] E. Perez, F. Strub, H. de Vries, V. Dumoulin, and A. C. Courville, "Film: Visual reasoning with a general conditioning layer," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, S. A. McIlraith and K. Q. Weinberger, Eds. AAAI Press, 2018, pp. 3942–3951.

[215] I. Krawczuk, P. Abranches, A. Loukas, and V. Cevher, "Gg-gan: A geometric graph generative adversarial network, 2021," in *URL https://openreview. net/forum*, 2020.

[216] N. D. Cao and T. Kipf, "Molgan: An implicit generative model for small molecular graphs," *CoRR*, vol. abs/1805.11973, 2018.

[217] H. Huang, L. Sun, B. Du, Y. Fu, and W. Lv, "Graphgdp: Generative diffusion processes for permutation invariant graph generation," in *IEEE International Conference on Data Mining, ICDM 2022, Orlando, FL, USA, November 28 - Dec. 1, 2022*, X. Zhu, S. Ranka, M. T. Thai, T. Washio, and X. Wu, Eds. IEEE, 2022, pp. 201–210.

[218] K. K. Haefeli, K. Martinkus, N. Perraudin, and R. Wattenhofer, "Diffusion models for graphs benefit from discrete state spaces," *CoRR*, vol. abs/2210.01549, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2210.01549

[219] X. Chen, J. He, X. Han, and L.-P. Liu, "Efficient and degree-guided graph generation via discrete diffusion modeling," in *Proceedings of the 40th International Conference on Machine Learning*, 2023, pp. 4585–4610.

[220] R. Gómez-Bombarelli, D. Duvenaud, J. M. Hernández-Lobato, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik, "Automatic chemical design using a data-driven continuous representation of molecules," *CoRR*, vol. abs/1610.02415, 2016.

[221] M. J. Kusner and J. M. Hernández-Lobato, "GANS for sequences of discrete elements with the gumbel-softmax distribution," *CoRR*, vol. abs/1611.04051, 2016.

[222] M. Simonovsky and N. Komodakis, "Graphvae: Towards generation of small graphs using variational autoencoders," in *Artificial Neural Networks and Machine Learning - ICANN 2018 - 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part I*, ser. Lecture Notes in Computer Science, V. Kurková, Y. Manolopoulos, B. Hammer, L. S. Iliadis, and I. Maglogiannis, Eds., vol. 11139. Springer, 2018, pp. 412–422.

[223] J. Mitton, H. M. Senn, K. Wynne, and R. Murray-Smith, "A graph VAE and graph transformer approach to generating molecular graphs," *CoRR*, vol. abs/2104.04345, 2021.

[224] L. Kong, J. Cui, H. Sun, Y. Zhuang, B. A. Prakash, and C. Zhang, "Autoregressive diffusion model for graph generation," in *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, ser. Proceedings of Machine Learning Research, A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, Eds., vol. 202. PMLR, 2023, pp. 17 391–17 408.

[225] R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. Von Lilienfeld, "Quantum chemistry structures and properties of 134 kilo molecules," *Scientific data*, vol. 1, no. 1, pp. 1–7, 2014.

[226] D. Polykovskiy, A. Zhebrak, B. Sánchez-Lengeling, S. Golovanov, O. Tatanov, S. Belyaev, R. Kurbanov, A. Artamonov, V. Aladinskiy, M. Veselov, A. Kadurin, S. I. Nikolenko, A. Aspuru-Guzik, and A. Zhavoronkov, "Molecular sets (MOSES): A benchmarking platform for molecular generation models," *CoRR*, vol. abs/1811.12823, 2018.

[227] N. Brown, M. Fiscato, M. H. S. Segler, and A. C. Vaucher, "Guacamol: Benchmarking models for de novo molecular design," *J. Chem. Inf. Model.*, vol. 59, no. 3, pp. 1096–1108, 2019.

[228] R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik, "Automatic chemical design using a data-driven continuous representation of molecules," *ACS central science*, vol. 4, no. 2, pp. 268–276, 2018.

[229] R. Mercado, T. Rastemo, E. Lindelöf, G. Klambauer, O. Engkvist, H. Chen, and E. J. Bjerrum, "Graph networks for molecular design," *Machine Learning: Science and Technology*, vol. 2, no. 2, p. 025023, 2021.

[230] M. H. Segler, T. Kogej, C. Tyrchan, and M. P. Waller, "Generating focused molecule libraries for drug discovery with recurrent neural networks," *ACS central science*, vol. 4, no. 1, pp. 120–131, 2018.

[231] Y. Kwon, D. Lee, Y.-S. Choi, K. Shin, and S. Kang, "Compressed graph representation for scalable molecular graph generation," *Journal of Cheminformatics*, vol. 12, pp. 1–8, 2020.

[232] J. H. Jensen, "A graph-based genetic algorithm and generative model/monte carlo tree search for the exploration of chemical space," *Chemical science*, vol. 10, no. 12, pp. 3567–3572, 2019.

[233] B. Jing, S. Gu, T. Chen, Z. Yang, D. Li, J. He, and K. Ren, "Towards editing time series," in *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, A. Globersons, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. M. Tomczak, and C. Zhang, Eds., 2024. [Online]. Available: http://papers.nips.cc/paper_files/paper/2024/hash/423d0909791493b7c10916fd328c2913-Abstract-Conference.html

[234] M. Xu, L. Yu, Y. Song, C. Shi, S. Ermon, and J. Tang, "Geodiff: A geometric diffusion model for molecular conformation generation," in *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. [Online]. Available: https://openreview.net/forum?id=PzcvxEMzvQC

[235] J. You, J. M. G. Selman, R. Ying, and J. Leskovec, "Identity-aware graph neural networks," in *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*. AAAI Press, 2021, pp. 10 737–10 745.

[236] K. Desai and J. Johnson, "Virtex: Learning visual representations from textual annotations," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*. Computer Vision Foundation / IEEE, 2021. [Online]. Available: https://openaccess.thecvf.com/content/CVPR2021/html/Desai_VirTex_Learning_Visual_Representations_From_Textual_Annotations_CVPR_2021_paper.html pp. 11 162–11 173.

[237] G. Mittal, J. H. Engel, C. Hawthorne, and I. Simon, "Symbolic music generation with diffusion models," in *Proceedings of the 22nd International Society for Music Information Retrieval Conference, ISMIR 2021, Online, November 7-12, 2021*, J. H. Lee, A. Lerch, Z. Duan, J. Nam, P. Rao, P. van Kranenburg, and A. Srinivasamurthy, Eds., 2021. [Online]. Available: https://archives.ismir.net/ismir2021/paper/000058.pdf pp. 468–475.

[238] J. Alayrac, J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson, K. Lenc, A. Mensch, K. Millican, M. Reynolds, R. Ring, E. Rutherford, S. Cabi, T. Han, Z. Gong, S. Samangooei, M. Monteiro, J. L. Menick, S. Borgeaud, A. Brock, A. Nematzadeh, S. Sharifzadeh, M. Binkowski, R. Barreira, O. Vinyals, A. Zisserman, and K. Simonyan, "Flamingo: a visual language model for few-shot learning," in *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., 2022. [Online]. Available: http://papers.nips.cc/paper_files/paper/2022/hash/960a172bc7fbf0177ccccbb411a7d800-Abstract-Conference.html

[239] H. Wang, S. Feng, T. He, Z. Tan, X. Han, and Y. Tsvetkov, "Can language models solve graph problems in natural language?" in *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., 2023. [Online]. Available: http://papers.nips.cc/paper_files/paper/2023/hash/622afc4edf2824a1b6aaf5afe153fa93-Abstract-Conference.html

[240] B. Fatemi, J. Halcrow, and B. Perozzi, "Talk like a graph: Encoding graphs for large language models," in *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. [Online]. Available: https://openreview.net/forum?id=IuXR1CCrSi

[241] R. Ye, C. Zhang, R. Wang, S. Xu, and Y. Zhang, "Natural language is all a graph needs," *CoRR*, vol. abs/2308.07134, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2308.07134

[242] R. Chen, T. Zhao, A. K. Jaiswal, N. Shah, and Z. Wang, "Llaga: Large language and graph assistant," in *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. [Online]. Available: https://openreview.net/forum?id=B48Pzc4oKi

[243] D. C. Zhang, M. Yang, R. Ying, and H. W. Lauw, "Text-attributed graph representation learning: Methods, applications, and challenges," in *Companion Proceedings of the ACM on Web Conference 2024, WWW 2024, Singapore, Singapore, May 13-17, 2024*, T. Chua, C. Ngo, R. K. Lee, R. Kumar, and H. W. Lauw, Eds. ACM, 2024. [Online]. Available: https://doi.org/10.1145/3589335.3641255 pp. 1298–1301.

[244] J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le, "Finetuned language models are zero-shot learners," in *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. [Online]. Available: https://openreview.net/forum?id=gEZrGCozdqR

[245] H. W. Chung, L. Hou, S. Longpre, B. Zoph, Y. Tay, W. Fedus, Y. Li, X. Wang, M. Dehghani, S. Brahma, A. Webson, S. S. Gu, Z. Dai, M. Suzgun, X. Chen, A. Chowdhery, A. Castro-Ros, M. Pellat, K. Robinson, D. Valter, S. Narang, G. Mishra, A. Yu, V. Y. Zhao, Y. Huang, A. M. Dai, H. Yu, S. Petrov, E. H. Chi, J. Dean, J. Devlin, A. Roberts, D. Zhou, Q. V. Le, and J. Wei, "Scaling instruction-finetuned language models," *J. Mach. Learn. Res.*, vol. 25, pp. 70:1–70:53, 2024. [Online]. Available: https://jmlr.org/papers/v25/23-0870.html

[246] Y. Li, H. Wen, W. Wang, X. Li, Y. Yuan, G. Liu, J. Liu, W. Xu, X. Wang, Y. Sun, R. Kong, Y. Wang, H. Geng, J. Luan, X. Jin, Z. Ye, G. Xiong, F. Zhang, X. Li, M. Xu, Z. Li, P. Li, Y. Liu, Y. Zhang, and Y. Liu, "Personal LLM agents: Insights and survey about the capability, efficiency and security," *CoRR*, vol. abs/2401.05459, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2401.05459

[247] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *J. Mach. Learn. Res.*, vol. 21, pp. 140:1–140:67, 2020. [Online]. Available: http://jmlr.org/papers/v21/20-074.html

[248] J. Huang, X. Zhang, Q. Mei, and J. Ma, "Can llms effectively leverage graph structural information: When and why," *CoRR*, vol. abs/2309.16595, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2309.16595

[249] D. Fu, Z. Hua, Y. Xie, J. Fang, S. Zhang, K. Sancak, H. Wu, A. Malevich, J. He, and B. Long, "Vcr-graphormer: A mini-batch graph transformer via virtual connections," in *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. [Online]. Available: https://openreview.net/forum?id=SUUrkC3STJ

[250] P. S. H. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html

[251] K. Guu, K. Lee, Z. Tung, P. Pasupat, and M. Chang, "Retrieval augmented language model pre-training," in *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, ser. Proceedings of Machine Learning Research, vol. 119. PMLR, 2020. [Online]. Available: http://proceedings.mlr.press/v119/guu20a.html pp. 3929–3938.

[252] G. Jeh and J. Widom, "Scaling personalized web search," in *Proceedings of the Twelfth International World Wide Web Conference, WWW 2003, Budapest, Hungary, May 20-24, 2003*, G. Hencsey, B. White, Y. R. Chen, L. Kovács, and S. Lawrence, Eds. ACM, 2003. [Online]. Available: https://doi.org/10.1145/775152.775191 pp. 271–279.

[253] J. Bromley, J. W. Bentz, L. Bottou, I. Guyon, Y. LeCun, C. Moore, E. Säckinger, and R. Shah, "Signature verification using A "siamese" time delay neural network," *Int. J. Pattern Recognit. Artif. Intell.*, vol. 7, no. 4, pp. 669–688, 1993. [Online]. Available: https://doi.org/10.1142/S0218001493000339

[254] B. Lester, R. Al-Rfou, and N. Constant, "The power of scale for parameter-efficient prompt tuning," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, M. Moens, X. Huang, L. Specia, and S. W. Yih, Eds. Association for Computational Linguistics, 2021. [Online]. Available: https://doi.org/10.18653/v1/2021.emnlp-main.243 pp. 3045–3059.

[255] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad, "Collective classification in network data," *AI Mag.*, vol. 29, no. 3, pp. 93–106, 2008. [Online]. Available: https://doi.org/10.1609/aimag.v29i3.2157

[256] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/5dd9db5e033da9c6fb5ba83c7a7ebea9-Abstract.html pp. 1024–1034.

[257] J. Chen, J. Zhu, and L. Song, "Stochastic training of graph convolutional networks with variance reduction," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, ser. Proceedings of Machine Learning Research, J. G. Dy and A. Krause, Eds., vol. 80. PMLR, 2018. [Online]. Available: http://proceedings.mlr.press/v80/chen18p.html pp. 941–949.

[258] M. Fey, J. E. Lenssen, F. Weichert, and J. Leskovec, "Gnnautoscale: Scalable and expressive graph neural networks via historical embeddings," in *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 2021. [Online]. Available: http://proceedings.mlr.press/v139/fey21a.html pp. 3294–3304.

[259] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=H1gL-2A9Ym

[260] R. Andersen, F. R. K. Chung, and K. J. Lang, "Local graph partitioning using pagerank vectors," in *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*. IEEE Computer Society, 2006. [Online]. Available: https://doi.org/10.1109/FOCS.2006.44 pp. 475–486.

[261] L. Liu, Z. Wang, R. Qiu, Y. Ban, E. Chan, Y. Song, J. He, and H. Tong, "Logic query of thoughts: Guiding large language models to answer complex logic queries with knowledge graphs," *CoRR*, vol. abs/2404.04264, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2404.04264

[262] J. Zou, D. Fu, S. Chen, X. He, Z. Li, Y. Zhu, J. Han, and J. He, "GTR: graph-table-rag for cross-table question answering," *CoRR*, vol. abs/2504.01346, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2504.01346

[263] M. Biehl, B. Hammer, and T. Villmann, "Prototype-based models in machine learning," *Wiley Interdisciplinary Reviews: Cognitive Science*, vol. 7, no. 2, pp. 92–111, 2016.

[264] N. Chen, Y. Li, J. Tang, and J. Li, "Graphwiz: An instruction-following language model for graph computational problems," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2024, Barcelona, Spain, August 25-29, 2024*, R. Baeza-Yates and F. Bonchi, Eds. ACM, 2024. [Online]. Available: https://doi.org/10.1145/3637528.3672010 pp. 353–364.

[265] W. Shi, S. Min, M. Yasunaga, M. Seo, R. James, M. Lewis, L. Zettlemoyer, and W. Yih, "REPLUG: retrieval-augmented black-box language models," *CoRR*, vol. abs/2301.12652, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2301.12652

[266] G. Izacard, P. Lewis, M. Lomeli, L. Hosseini, F. Petroni, T. Schick, J. Dwivedi-Yu, A. Joulin, S. Riedel, and E. Grave, "Atlas: Few-shot learning with retrieval augmented language models," *Journal of Machine Learning Research*, vol. 24, no. 251, pp. 1–43, 2023.

[267] X. He, X. Bresson, T. Laurent, A. Perold, Y. LeCun, and B. Hooi, "Harnessing explanations: Llm-to-lm interpreter for enhanced text-attributed graph representation learning," in *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. [Online]. Available: https://openreview.net/forum?id=RXFVcynVe1

[268] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," *CoRR*, vol. abs/2005.00687, 2020.

[269] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, "Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers," *Advances in Neural Information Processing Systems*, vol. 33, pp. 5776–5788, 2020.

[270] H. W. Chung, L. Hou, S. Longpre, B. Zoph, Y. Tay, W. Fedus, E. Li, X. Wang, M. Dehghani, S. Brahma, A. Webson, S. S. Gu, Z. Dai, M. Suzgun, X. Chen, A. Chowdhery, S. Narang, G. Mishra, A. Yu, V. Y. Zhao, Y. Huang, A. M. Dai, H. Yu, S. Petrov, E. H. Chi, J. Dean, J. Devlin, A. Roberts, D. Zhou, Q. V. Le, and J. Wei, "Scaling instruction-finetuned language models," *CoRR*, vol. abs/2210.11416, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2210.11416

[271] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=SJU4ayYgl

[272] M. He, Z. Wei, Z. Huang, and H. Xu, "Bernnet: Learning arbitrary graph spectral filters via bernstein approximation," in *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021. [Online]. Available: https://proceedings.neurips.cc/paper/2021/hash/76f1cfd7754a6e4fc3281bcccb3d0902-Abstract.html pp. 14 239–14 251.

[273] D. Bo, X. Wang, C. Shi, and H. Shen, "Beyond low-frequency information in graph convolutional networks," in *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*. AAAI Press, 2021. [Online]. Available: https://doi.org/10.1609/aaai.v35i5.16514 pp. 3950–3957.

[274] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," in *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, ser. Proceedings of Machine Learning Research, vol. 119. PMLR, 2020. [Online]. Available: http://proceedings.mlr.press/v119/chen20v.html pp. 1725–1735.

[275] S. Luan, C. Hua, Q. Lu, J. Zhu, M. Zhao, S. Zhang, X. Chang, and D. Precup, "Revisiting heterophily for graph neural networks," in *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., 2022. [Online]. Available: http://papers.nips.cc/paper_files/paper/2022/hash/092359ce5cf60a80e882378944bf1be4-Abstract-Conference.html

[276] J. Zhao, M. Qu, C. Li, H. Yan, Q. Liu, R. Li, X. Xie, and J. Tang, "Learning on large-scale text-attributed graphs via variational inference," in *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. [Online]. Available: https://openreview.net/forum?id=q0nmYciuuZN

[277] E. Chien, W. Chang, C. Hsieh, H. Yu, J. Zhang, O. Milenkovic, and I. S. Dhillon, "Node feature extraction by self-supervised multi-scale neighborhood prediction," in *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. [Online]. Available: https://openreview.net/forum?id=KJggliHbs8

[278] G. Li, M. Müller, B. Ghanem, and V. Koltun, "Training graph neural networks with 1000 layers," in *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 2021. [Online]. Available: http://proceedings.mlr.press/v139/li21o.html pp. 6437–6449.

[279] P. He, X. Liu, J. Gao, and W. Chen, "Deberta: decoding-enhanced bert with disentangled attention," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. [Online]. Available: https://openreview.net/forum?id=XPZIaotutsD

[280] Q. Li, Z. Han, and X. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, S. A. McIlraith and K. Q. Weinberger, Eds. AAAI Press, 2018. [Online]. Available: https://doi.org/10.1609/aaai.v32i1.11604 pp. 3538–3545.

[281] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *CoRR*, vol. abs/1611.07308, 2016. [Online]. Available: http://arxiv.org/abs/1611.07308

[282] K. Duan, Q. Liu, T.-S. Chua, S. Yan, W. T. Ooi, Q. Xie, and J. He, "Simteg: A frustratingly simple approach improves textual graph learning," *arXiv preprint arXiv:2308.02565*, 2023.

[283] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *CoRR*, vol. abs/1503.02531, 2015. [Online]. Available: http://arxiv.org/abs/1503.02531

[284] S. Zhang, D. Zhou, M. Y. Yildirim, S. Alcorn, J. He, H. Davulcu, and H. Tong, "Hidden: hierarchical dense subgraph detection with application to financial fraud detection," in *SDM*. SIAM, 2017, pp. 570–578.

[285] D. Jiang, Z. Wu, C.-Y. Hsieh, G. Chen, B. Liao, Z. Wang, C. Shen, D. Cao, J. Wu, and T. Hou, "Could graph neural networks learn better molecular representation for drug discovery? a comparison study of descriptor-based and graph-based models," *Journal of cheminformatics*, vol. 13, no. 1, pp. 1–23, 2021.

[286] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann, "Tudataset: A collection of benchmark datasets for learning with graphs," in *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020. [Online]. Available: www.graphlearning.io

[287] T. Wang, J.-Y. Zhu, A. Torralba, and A. A. Efros, "Dataset distillation," *arXiv preprint arXiv:1811.10959*, 2018.

[288] G. Cazenavette, T. Wang, A. Torralba, A. A. Efros, and J.-Y. Zhu, "Dataset distillation by matching training trajectories," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 4750–4759.

[289] B. Zhao, K. R. Mopuri, and H. Bilen, "Dataset condensation with gradient matching." *ICLR*, vol. 1, no. 2, p. 3, 2021.

[290] B. Colson, P. Marcotte, and G. Savard, "Bilevel programming: A survey," *4or*, vol. 3, no. 2, pp. 87–107, 2005.

[291] S. S. Du, K. Hou, R. R. Salakhutdinov, B. Poczos, R. Wang, and K. Xu, "Graph neural tangent kernel: Fusing graph neural networks with graph kernels," *Advances in neural information processing systems*, vol. 32, 2019.

[292] H. Kashima, K. Tsuda, and A. Inokuchi, "Marginalized kernels between labeled graphs," in *Proceedings of the 20th international conference on machine learning (ICML-03)*, 2003, pp. 321–328.

[293] U. Kang, H. Tong, and J. Sun, "Fast random walk graph kernel," in *Proceedings of the 2012 SIAM international conference on data mining*. SIAM, 2012, pp. 828–838.

[294] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=S1jE5L5gl

[295] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=rkE3y85ee

[296] K. Fukushima, "Cognitron: A self-organizing multilayered neural network," *Biological cybernetics*, vol. 20, no. 3, pp. 121–136, 1975.

[297] K. Tieu, D. Fu, Y. Zhu, H. F. Hamann, and J. He, "Temporal graph neural tangent kernel with graphon-guaranteed," in *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, A. Globersons, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. M. Tomczak, and C. Zhang, Eds., 2024. [Online]. Available: http://papers.nips.cc/paper_files/paper/2024/hash/abd3c6b90e474ec50a52c446926b00be-Abstract-Conference.html

[298] S. Arora, S. S. Du, W. Hu, Z. Li, R. R. Salakhutdinov, and R. Wang, "On exact computation with an infinitely wide neural net," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

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

[300] F. Pedregosa, "Hyperparameter optimization with approximate gradient," in *International conference on machine learning*. PMLR, 2016, pp. 737–746.

[301] J. Lorraine, P. Vicol, and D. Duvenaud, "Optimizing millions of hyperparameters by implicit differentiation," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020, pp. 1540–1552.

[302] D. Maclaurin, D. Duvenaud, and R. Adams, "Gradient-based hyperparameter optimization through reversible learning," in *International conference on machine learning*. PMLR, 2015, pp. 2113–2122.

[303] T. Nguyen, Z. Chen, and J. Lee, "Dataset meta-learning from kernel ridge-regression," in *International Conference on Learning Representations*, 2021.

[304] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[305] J. Gasteiger, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," in *International Conference on Learning Representations*, 2019.

[306] S. Suresh, P. Li, C. Hao, and J. Neville, "Adversarial graph augmentation to improve graph contrastive learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 15 920–15 933, 2021.

[307] M. Welling, "Herding dynamical weights to learn," in *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009, pp. 1121–1128.

[308] O. Sener and S. Savarese, "Active learning for convolutional neural networks: A core-set approach," in *International Conference on Learning Representations*, 2018.

[309] R. Z. Farahani and M. Hekmatfar, *Facility location: concepts, models, algorithms and case studies*. Springer Science & Business Media, 2009.

[310] T. F. Gonzalez, "Clustering to minimize the maximum intercluster distance," *Theoretical computer science*, vol. 38, pp. 293–306, 1985.

[311] A. Abid, M. F. Balin, and J. Zou, "Concrete autoencoders for differentiable feature selection and reconstruction," *arXiv preprint arXiv:1901.09346*, 2019.

[312] S. Chang, Y. Zhang, M. Yu, and T. S. Jaakkola, "Invariant rationalization," in *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, ser. Proceedings of Machine Learning Research, vol. 119. PMLR, 2020, pp. 1448–1458.

[313] G. Liu, T. Zhao, J. Xu, T. Luo, and M. Jiang, "Graph rationalization with environment-based augmentations," in *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14 - 18, 2022*, A. Zhang and H. Rangwala, Eds. ACM, 2022, pp. 1069–1078.

[314] Y. Wu, X. Wang, A. Zhang, X. He, and T. Chua, "Discovering invariant rationales for graph neural networks," in *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

[315] Z. Zhang, B. Du, and H. Tong, "Suger: A subgraph-based graph convolutional network method for bundle recommendation," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management, Atlanta, GA, USA, October 17-21, 2022*, M. A. Hasan and L. Xiong, Eds. ACM, 2022, pp. 4712–4716.

[316] L. Yue, Q. Liu, Y. Liu, W. Gao, F. Yao, and W. Li, "Cooperative classification and rationalization for graph generalization," in *Proceedings of the ACM on Web Conference 2024, WWW 2024, Singapore, May 13-17, 2024*, T. Chua, C. Ngo, R. Kumar, H. W. Lauw, and R. K. Lee, Eds. ACM, 2024. [Online]. Available: https://doi.org/10.1145/3589334.3645332 pp. 344–352.

[317] Y. Sui, Q. Wu, J. Wu, Q. Cui, L. Li, J. Zhou, X. Wang, and X. He, "Unleashing the power of graph data augmentation on covariate distribution shift," in *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., 2023. [Online]. Available: http://papers.nips.cc/paper_files/paper/2023/hash/3a33ddacb2798fc7d83b8334d552e05a-Abstract-Conference.html

[318] J. Kaddour, A. Lynch, Q. Liu, M. J. Kusner, and R. Silva, "Causal machine learning: A survey and open problems," *CoRR*, vol. abs/2206.15475, 2022.

[319] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017, pp. 5998–6008.

[320] R. Qiu, Z. Xu, W. Bao, and H. Tong, "Ask, and it shall be given: On the turing completeness of prompting," in *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. [Online]. Available: https://openreview.net/forum?id=AS8SPTyBgw

[321] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma, "Linformer: Self-attention with linear complexity," *CoRR*, vol. abs/2006.04768, 2020.

[322] L. J. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *CoRR*, vol. abs/1607.06450, 2016.

[323] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, ser. JMLR Workshop and Conference Proceedings, F. R. Bach and D. M. Blei, Eds., vol. 37. JMLR.org, 2015, pp. 448–456.

[324] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[325] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 32, no. 1, pp. 4–24, 2021.

[326] D. Chen, L. O'Bray, and K. M. Borgwardt, "Structure-aware transformer for graph representation learning," in *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu, and S. Sabato, Eds., vol. 162. PMLR, 2022, pp. 3469–3489.

[327] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontañón, P. Pham, A. Ravula, Q. Wang, L. Yang, and A. Ahmed, "Big bird: Transformers for longer sequences," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020.

[328] V. P. Dwivedi, C. K. Joshi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," *J. Mach. Learn. Res.*, vol. 24, pp. 43:1–43:48, 2023.

[329] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *ICLR*, 2019.

[330] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[331] S. Brody, U. Alon, and E. Yahav, "How attentive are graph attention networks?" in *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

[332] X. Bresson and T. Laurent, "Residual gated graph convnets," *CoRR*, vol. abs/1711.07553, 2017.

[333] V. P. Dwivedi and X. Bresson, "A generalization of transformer networks to graphs," *CoRR*, vol. abs/2012.09699, 2020.

[334] G. Mialon, D. Chen, M. Selosse, and J. Mairal, "Graphit: Encoding graph structure in transformers," *CoRR*, vol. abs/2106.05667, 2021.

[335] D. Kreuzer, D. Beaini, W. L. Hamilton, V. Létourneau, and P. Tossou, "Rethinking graph transformers with spectral attention," in *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021, pp. 21 618–21 629.

[336] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T. Liu, "Do transformers really perform badly for graph representation?" in *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021, pp. 28 877–28 888.

[337] Z. Wu, P. Jain, M. A. Wright, A. Mirhoseini, J. E. Gonzalez, and I. Stoica, "Representing long-range context for graph neural networks with global attention," *CoRR*, vol. abs/2201.08821, 2022.

[338] L. Rampásek, M. Galkin, V. P. Dwivedi, A. T. Luu, G. Wolf, and D. Beaini, "Recipe for a general, powerful, scalable graph transformer," in *NeurIPS*, 2022.

[339] J. Huang, X. Zhang, Q. Mei, and J. Ma, "Can llms effectively leverage graph structural information through prompts, and why?" *Trans. Mach. Learn. Res.*, vol. 2024, 2024. [Online]. Available: https://openreview.net/forum?id=L2jRavXRxs

[340] A. Lou, C. Meng, and S. Ermon, "Discrete diffusion modeling by estimating the ratios of the data distribution," in *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. [Online]. Available: https://openreview.net/forum?id=CNicRIVIPA

[341] K. Tian, Y. Jiang, Z. Yuan, B. Peng, and L. Wang, "Visual autoregressive modeling: Scalable image generation via next-scale prediction," in *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, A. Globersons, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. M. Tomczak, and C. Zhang, Eds., 2024. [Online]. Available: http://papers.nips.cc/paper_files/paper/2024/hash/9a24e284b187f662681440ba15c416fb-Abstract-Conference.html