

Copyright Undertaking

This thesis is protected by copyright, with all rights reserved.

By reading and using the thesis, the reader understands and agrees to the following terms:

1. The reader will abide by the rules and legal ordinances governing copyright regarding the use of the thesis.
2. The reader will use the thesis for the purpose of research or private study only and not for distribution or further reproduction or any other purpose.
3. The reader agrees to indemnify and hold the University harmless from and against any loss, damage, cost, liability or expenses arising from copyright infringement or unauthorized usage.

IMPORTANT

If you have reasons to believe that any materials in this thesis are deemed not suitable to be distributed in this form, or a copyright owner having difficulty with the material being included in our database, please contact lbsys@polyu.edu.hk providing details. The Library will look into your claim and consider taking remedial action upon receipt of the written requests.

TOWARDS PRACTICAL GRAPH NEURAL
NETWORKS: ADDRESSING LABEL
SCARCITY, HETEROGENEITY, AND
INFERENCE EFFICIENCY

ZIANG ZHOU

PhD

The Hong Kong Polytechnic University

2026

The Hong Kong Polytechnic University

Department of Computing

Towards Practical Graph Neural Networks: Addressing Label
Scarcity, Heterogeneity, and Inference Efficiency

Ziang Zhou

A thesis submitted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy

August 2025

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgment has been made in the text.

Signature: _____

Name of Student: Ziang Zhou

Abstract

Graph Neural Networks (GNNs) have become a cornerstone for learning on graph-structured data, yet their widespread adoption in real-world scenarios is often impeded by significant practical challenges. This thesis aims to bridge the gap between the theoretical power of GNNs and their practical applicability by systematically addressing three critical limitations: data scarcity, structural heterogeneity, and model inefficiency.

First, we confront the challenge of data scarcity, where GNNs often falter when extremely few labeled data is available. To overcome this, we introduce a stabilized self-training framework. This method enhances model performance in few-labeled settings by dynamically generating high-quality pseudo-labels in a robust and balanced manner, thereby stabilizing the training process and significantly improving predictive accuracy with limited supervision.

Moving from data limitations to structural complexity, we tackle the prevalent issue of heterogeneous graphs, which contain diverse types of nodes and edges. We identify that conventional GNNs often entangle distinct relational semantics during message passing. In response, we propose SlotGAT, a novel architecture employing a slot-based message passing mechanism. By maintaining separate information channels, or *slots*, for each node type, SlotGAT effectively preserves rich, type-specific semantics, leading to more expressive and accurate graph representations.

Finally, to address the challenge of model efficiency and deployment constraints,

we present TINED, a novel GNN-to-MLP knowledge distillation framework. Rather than merely matching output distributions, TINED performs a fine-grained, layer-wise transfer of knowledge. It achieves this through two key innovations: *Teacher Injection*, which directly transplants compatible parameters from the GNN teacher to the MLP student, and *Dirichlet Energy Distillation*, which ensures the student model emulates the specific smoothing properties of each layer in the teacher. This results in a lightweight, fast MLP that retains the strong performance of its GNN counterpart without the costly message-passing operations.

Collectively, these contributions provide a suite of solutions that enhance the robustness, expressiveness, and efficiency of GNNs. By making these models more stable with scarce data, more capable on complex heterogeneous structures, and more feasible for practical deployment, this thesis pushes GNNs towards being more broadly applicable and impactful tools for real-world problems.

Publications Arising from the Thesis

1. Ziang Zhou, Jieming Shi, Shengzhong Zhang, Zengfeng Huang, Qing Li, “Effective Stabilized Self-Training on Few-Labeled Graph Data”, in *Information Sciences (INS)* (2023).
2. Ziang Zhou, Jieming Shi, Renchi Yang, Yuanhang Zou, and Qing Li, “Slot-GAT: Slot-based Message Passing for Heterogeneous Graphs”, in *International Conference on Machine Learning (ICML)* (2023).
3. Ziang Zhou, Zhihao Ding, Jieming Shi, Qing Li, and Shiqi Shen, “TINED: GNNs-to-MLPs by Teacher Injection and Dirichlet Energy Distillation”, in *International Conference on Machine Learning (ICML)* (2025).

Dedication

To my family.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my chief supervisor, Prof. Jieming Shi and co-supervisor, Prof. Qing Li, for their unwavering guidance, endless patience, and invaluable support throughout my doctoral journey. Their profound knowledge, insightful feedback, and encouraging mentorship have been instrumental in shaping both my research and my growth as a scholar. I am truly fortunate to have had the opportunity to work under their supervision.

I extend my sincere appreciation to the members of my thesis examination committee and all the professors who have assisted me through my research journey, including but not limited to Prof. Zengfeng Huang, Prof. Shengcai Liu, Prof. Haishan Ye, Prof. Tong Zhang (in alphabetical order). Their profound expertise, rigorous academic standards, and constructive feedback have significantly elevated the quality of this work. I am deeply honored to have benefited from their wisdom and insights, which have not only strengthened my research but also broadened my academic perspective.

Finally, I would like to thank my friends and labmates, both in Hong Kong and elsewhere, who have enriched my doctoral journey through countless hours of intellectual discussions, shared experiences, and joyful moments. Special thanks to Dr. Rengjie Zhang, Dr. Xin Wang, Dr. Jiaying Wen, Dr. Huang Zheng, Dr. Hanze Dong, Dr. Yikai Wang, Dr. Jianyi Zhang, Dr. Rengtian Yao, Dr. Yiran Li, Dr. Wei Tian, Dr. Zhihao Ding, Xinle Chen, Dr. Haicang Zhou, Dr. Shengzhong Zhang, Dr. Zhaoyang Xia, Dr. Yilang Zhang, Dr. Mengchu Xu, Hanwen Wu, Zuyuan Zhong, Lvxing Xue,

Boyu Fu, Tongshu Ding, Jinxuan Fan (in random order). Their friendship, encouragement, and companionship have made my time in Hong Kong truly memorable and this challenging journey remarkably rewarding.

I am deeply grateful to my parents for their unconditional love, unwavering support, and continuous encouragement throughout my academic journey. Their sacrifices, wisdom, and faith in my abilities have been the foundation of my achievements. I also want to acknowledge our beloved family dog, Beibei, whose joyful presence and loyal companionship have brought countless moments of happiness and comfort to our family.

Most importantly, I owe my deepest gratitude to my wife Nina, whose love, patience, and understanding have been my greatest source of strength. Her unwavering support during the most challenging times, particularly during the isolation periods of the COVID-19 pandemic in my first and second years of doctoral studies, will remain etched in my memory forever. Those difficult times, when we could only rely on each other, forged an unbreakable bond that sustained me through this entire journey. I am also blessed to welcome our precious daughter during my fifth year of studies, whose arrival has filled our lives with immeasurable joy and given new meaning to my pursuits. To my wife and daughter, thank you for being my inspiration and my home.

Any errors or omissions in this thesis remain entirely my own responsibility.

Ziang Zhou

Hong Kong

June 2025

Table of Contents

Abstract	i
Publications Arising from the Thesis	iii
Dedication	iv
Acknowledgments	v
List of Figures	xii
List of Tables	xv
1 Introduction	1
1.1 Background and Motivation	1
1.1.1 Graphs	1
1.1.2 Graph Neural Networks	4
1.2 Problem Statement and Core Challenges	5
1.2.1 Data Scarcity in Few-Labeled Graph Learning	5
1.2.2 Structural Complexity in Heterogeneous Graph Modeling . . .	6

1.2.3	Model Efficiency and Deployment Scalability	7
1.2.4	Summary of Challenges	8
1.3	Research Questions and Objectives	10
1.4	Main Contributions	10
1.5	Thesis Outline	12
2	Literature Review	14
2.1	Introduction to the Chapter	14
2.2	Fundamentals of Graph Neural Networks	14
2.3	Learning on Graphs with Few Labels	19
2.4	Graph Neural Networks for Heterogeneous Graphs	20
2.5	Enhancing Model Efficiency and Deployment	21
2.6	Summary and Research Gaps	22
3	Effective Stabilized Self-Training on Few-Labeled Graph Data	24
3.1	Introduction	24
3.2	Related Work	28
3.3	Preliminaries	31
3.4	Our Method and Analysis	33
3.4.1	Empirical Analysis	33
3.4.2	Theoretical Analysis	35
3.4.3	Stabilized Self-Training (SST) Framework	38
3.5	Experiments	42

3.5.1	Implementation Details	43
3.5.2	Datasets and Competitors	44
3.5.3	Experimental Settings	45
3.5.4	Overall Experimental Results	47
3.5.5	Ablation Study	50
3.6	Conclusion	53
3.7	Proof	53
3.7.1	Proof of Eq. 3.5	54
3.7.2	Proof of Lemma 1	55
4	SlotGAT: Slot-based Message Passing for Heterogeneous Graphs	58
4.1	Introduction	58
4.2	Preliminary	61
4.3	Related Work	62
4.4	The SlotGAT Method	64
4.4.1	Node Type Slot Initialization	65
4.4.2	Slot-based Message Passing Layer	66
4.4.3	Slot Attention	68
4.4.4	Downstream Objectives	70
4.4.5	Analysis	72
4.4.6	Proof of Theorem 1	73
4.5	Experiments	76

4.5.1	Experiment Settings	76
4.5.2	Searched Hyper-parameters	83
4.5.3	Evaluation Results	84
4.5.4	Model Analysis	85
4.6	Conclusion, Limitation, and Future Work	89
4.6.1	Additional Visualization Results	90
5	TINED: GNNs-to-MLPs by Teacher Injection and Dirichlet Energy Distillation	91
5.1	Introduction	91
5.2	Related Work	94
5.3	Preliminaries	95
5.4	The TINED Method	97
5.4.1	Teacher Injection	98
5.4.2	Proof of Theorem 2	99
5.4.3	Dirichlet Energy Distillation	101
5.4.4	Discussion on Why to Choose Dirichlet Energy	103
5.5	Experiments	103
5.5.1	Datasets	104
5.5.2	Experiment Settings	104
5.5.3	Teacher details and hyperparameter search space	107
5.5.4	Performance in Transductive Setting	108
5.5.5	Performance in Production Setting with <i>ind</i> & <i>tran</i>	109

5.5.6	Inference Time	112
5.5.7	Different Teacher GNN Architectures	113
5.5.8	Experimental Analysis	115
5.5.9	Layer-wise t-SNE Visualization	116
5.6	Conclusion	124
6	Conclusion	125
6.1	Summary of Contributions	125
6.1.1	Effective Learning with Limited Labels	125
6.1.2	Advancing Heterogeneous Graph Modeling	126
6.1.3	Efficient GNN Deployment through Knowledge Distillation . .	127
6.2	Broader Impact and Significance	127
6.3	Limitations and Future Directions	128
6.3.1	Current Limitations and Future Directions	128
6.4	Concluding Remarks	129
	References	131

List of Figures

1.1	Example of basic social network. This network represents friendships where nodes are individuals and edges are friendship relationships. The vector under each node represents the node’s features.	2
1.2	Heterogeneous Citation Network. The graph shows a citation network with three types of nodes: papers, authors, and venues. Different edge types represent various relationships: citation connections (blue lines), author-paper connections (red lines), and field-paper connections (green lines).	3
1.3	Challenge of few-labeled graph learning with only 1-2 labeled nodes per class (colored nodes).	5
1.4	Overview of the framework of this thesis. We start with the basic concepts of graph neural networks and then introduce the three main challenges in GNN applicability and our proposed solutions: (A) Data scarcity challenge and the Stabilized Self-Training framework, (B) Heterogeneous graph complexity challenge and the SlotGAT architecture, and (C) Model efficiency challenge and the TINED distillation method. The framework is designed to address the challenges of data scarcity, structural complexity, and model efficiency in graph neural networks, and to make GNNs more applicable, robust, and efficient for a wider range of real-world challenges.	9

3.1	The distribution of predicted labels in different classes in Cora-1. . .	25
3.2	Training GCN on Cora with 1 labeled node per class (Cora-1)	34
3.3	Confidence distributions of GCN Training on Cora-1.	34
3.4	Ablation study of SST on Cora.	51
3.5	Vary β . Results are averaged over 20 runs.	52
4.1	(a) A heterogeneous graph. (b) The slot-based message passing in the proposed SlotGAT: every node has 3 slots, corresponding to 3 node-type feature spaces. For example, in the input layer, v_2 's slot 1 is initialized by its features since v_2 is in type 1, while slots 0 and 2 of v_2 are empty. The message passing in SlotGAT is slot specific (colored dashed arrows), <i>e.g.</i> , neighboring slot messages passed to the slots of the same node type on node v_2 in Layer 1. (c) The semantic mixing issue: every node maintains a single representation; in Layer 1, v_2 aggregates message from v_1 by firstly applying transformation $\tau(0, 1)$ to convert v_1 's representation in type 0 to the feature space of v_2 in type 1, which mixes the two feature spaces.	61
4.2	The SlotGAT architecture. (i) SlotGAT initializes every node with multiple slots corresponding to all node types (3 types in the example), and the slot for the node's type is initialized by its transformed features, while the other slots are empty. (ii) In a slot-based message passing layer, slot representations are transformed and propagated separately, with an attention based aggregation technique. (iii) After the last L -th layer, SlotGAT includes a slot attention technique to integrate slots for downstream tasks.	63
4.3	Attention mechanism in a message passing layer.	67

4.4	Slot attention	69
4.5	Tsne visualization on all slot representations of 2-nd SlotGAT layer on IMDB	85
4.6	Tsne visualization on all slot representations of 3-rd SlotGAT layer on DBLP	89
5.1	The DE ratios of FTs and GPs in the layers of GraphSAGE.	92
5.2	(a) TINED with Teacher Injection and Dirichlet Energy Distillation; (b) Inference settings	95
5.3	Inference Time and Accuracy	112
5.4	Accuracy on Different Teacher GNNs	112
5.5	The layer-wise DE ratio of trained GCN, GraphSAGE and GAT, with 2 layers architecture on different datasets. The x -axis is the layer number, and the y -axis is the DE ratio value.	114
5.6	t-SNE of model embeddings at different training stages on Citeseer. (a) SAGE (teacher model at epoch 500), (b, c) TINED (epochs 0 and 500), (d, e) GLNN* (epochs 0 and 500).	116
5.7	Layer-wise t-SNE visualization of the embeddings for various models (GraphSAGE, TINED, GLNN*) and stages (epoch 0 at initialization and epoch 500 after training convergence) on Citeseer.	117
5.8	Learned DE ratios in MLPs v.s. ground truth of 2-layer GraphSAGE.	118
5.9	The layer-wise DE ratio of trained GCN, GraphSAGE and GAT, with 5 layers architecture on different datasets. The x -axis is the layer number, and the y -axis is the DE ratio value.	119

List of Tables

3.1	Dataset Statistics	43
3.2	Absolute accuracy improvements (in percentage) of SST (<i>i.e.</i> , SST-GCN and SSTDA) over base models GCN and DAGNN on 4 datasets, averaged over 100 random data splits.	45
3.3	Accuracy results (in percentage) on PubMed and Cora-full respectively, averaged over 100 random data splits. (The best accuracy is in bold .)	46
3.4	Accuracy results (in percentage) on Cora and CiteSeer respectively, averaged over 100 random data splits. (The best accuracy is in bold .)	47
3.5	Accuracy results (in percentage) on Ogbn-arxiv, averaged over 100 random data splits. (The best accuracy is in bold .)	49
3.6	Total training time in seconds (s) and training time per epoch in milliseconds (ms) on Ogbn-arxiv.	50
3.7	The number of pseudo labels when varying β in SSTGCN, averaged over 20 runs.	52
3.8	The number of pseudo labels when varying β in SSTDA, averaged over 20 runs.	53
4.1	Statistics of Benchmark Datasets.	75

4.2	Node classification results with mean and standard deviation of Macro-F1/Micro-F1 (Part I). Vacant positions (" ") mean out of memory. Best is in bold, and runner up is underlined.	76
4.3	Node classification results with mean and standard deviation of Macro-F1/Micro-F1 (Part II). Vacant positions (" ") mean out of memory. Best is in bold, and runner up is underlined.	77
4.4	Data Split of Node Classification Datasets.	78
4.5	The searched hyper-parameters for SlotGAT with best performances on various datasets (Part 1).	78
4.6	The searched hyper-parameters for SlotGAT with best performances on various datasets (Part 2).	78
4.7	Meta-paths used in our experiments.	81
4.8	Link prediction results with mean and standard deviation of ROC-AUC/MRR. Vacant positions (" ") means out of memory.	83
4.9	Statistical Significance Test (Node Classification)	85
4.10	Statistical Significance Test (Link Prediction)	85
4.11	Ablation for Slot Attention (Macro-F1/Micro-F1) on node classification. Best is in bold, and runner up is underlined.	86
4.12	Statistical Significance Test between SlotGAT and its variants w.r.t. Micro-F1.	86
4.13	Statistical Significance Test between SlotGAT and its variants w.r.t. Macro-F1.	87
4.14	Training time per epoch (millisecond)	88
4.15	Inference time (millisecond)	88

4.16	Peak GPU memory (GB)	88
5.1	Dataset Statistics and Splits.	104
5.2	Under the transductive(<i>tran</i>) setting, the accuracy results of the teacher and methods without graph dependency for inference on $\mathcal{V}^U$ (higher is better). The best is in bold. OOM means out-of-memory.	105
5.3	Under the transductive(<i>tran</i>) setting, the accuracy results of the teacher and methods with graph dependency for inference on $\mathcal{V}^U$ (higher is better). The best is in bold.	105
5.4	Hyperparameters for GNNs on five datasets from [159, 116].	107
5.5	In <i>prod</i> (<i>ind</i> & <i>tran</i>) setting, the accuracy results of the teacher and all methods without graph dependency for online inference are presented. <i>ind</i> indicates the results on $\mathcal{V}_{ind}^U$, <i>tran</i> indicates the results on $\mathcal{V}_{obs}^U$, and <i>prod</i> indicates the weighted sum of the performance of both <i>ind</i> and <i>tran</i> with weights proportional to the size of $\mathcal{V}_{ind}^U$ and $\mathcal{V}_{obs}^U$. The best result in each category is in bold. The teacher’s performance surpassing the best result in at least one category is italicized.	110
5.6	In <i>prod</i> (<i>ind</i> & <i>tran</i>) setting, the accuracy results of the teacher and all methods with graph dependency for online inference are presented. <i>ind</i> indicates the results on $\mathcal{V}_{ind}^U$, <i>tran</i> indicates the results on $\mathcal{V}_{obs}^U$, and <i>prod</i> indicates the weighted sum of the performance of both <i>ind</i> and <i>tran</i> with weights proportional to the size of $\mathcal{V}_{ind}^U$ and $\mathcal{V}_{obs}^U$. The best result in each category is in bold. The teacher’s performance surpassing the best result in at least one category is italicized.	111
5.7	Ablation study	115
5.8	Approximation error	120

5.9	Vary β of TINED on transductive setting. Best results are in bold.	120
5.10	Vary η of TINED on transductive setting. Best results are in bold.	121
5.11	Vary ζ for TINED on large graph ogbn-arxiv under transductive setting.	121
5.12	Experiment results under <i>prod</i> (<i>ind</i> & <i>tran</i>) setting on heterophilic datasets - Methods without graph dependency. The best result in each category is in bold.	122
5.13	Experiment results under <i>prod</i> (<i>ind</i> & <i>tran</i>) setting on heterophilic datasets - Methods with graph dependency. The best result in each category is in bold.	122
5.14	Compare with MLP* with 4 layers under transductive setting.	123
5.15	Comparison with MLP* with 4 layers under <i>prod(ind&tran)</i> setting	123

Chapter 1

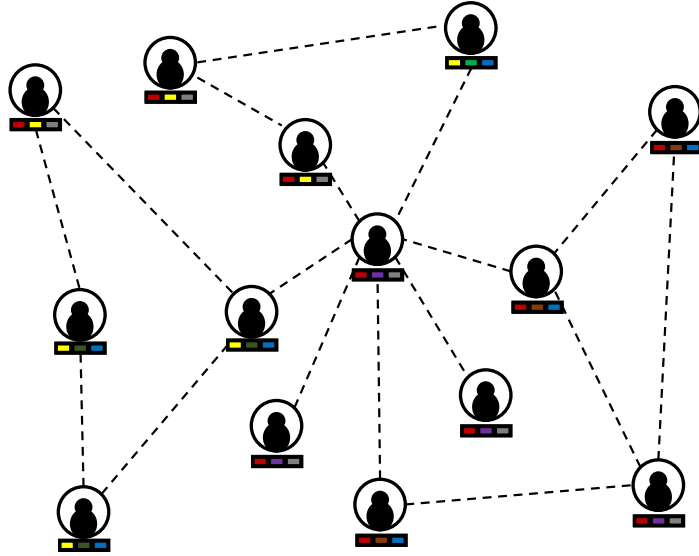
Introduction

1.1 Background and Motivation

1.1.1 Graphs

Entities and relationships between entities are the fundamental building blocks of the world. Graph is a mathematical framework that captures these relationships in a structured way. Formally, a graph $G = (V, E)$ is defined as a collection of vertices $V = \{v_1, v_2, \dots, v_n\}$ and edges $E \subseteq V \times V$ that connect pairs of vertices [158]. This foundational structure can be enriched with additional information to capture different types of real-world relationships. In attributed graphs $G = (V, E, X)$, each vertex v_i is associated with a feature vector $x_i \in \mathbb{R}^d$, forming the feature matrix $X \in \mathbb{R}^{n \times d}$ [47]. Directed graphs introduce asymmetric relationships where $E \subseteq V \times V$ represents ordered pairs. Heterogeneous graphs further extend this framework with typed vertices and edges: $G = (V, E, \mathcal{A}, \mathcal{R})$, where $\mathcal{A}$ and $\mathcal{R}$ denote vertex and edge type sets respectively, and each vertex $v \in V$ has type $\tau(v) \in \mathcal{A}$ while each edge $e \in E$ has type $\phi(e) \in \mathcal{R}$ [32, 56]. This mathematical framework provides an elegant means to model complex relational structures across diverse domains. The key insight lies in

mapping entities to vertices and their relationships to edges, transforming irregular, non-Euclidean data into structured mathematical objects amenable to computational analysis [3]. Such representations naturally preserve both local connectivity patterns and global structural characteristics, enabling algorithms to leverage topological and relational properties of the underlying system [9, 26].



(A) Social Network

Figure 1.1: Example of basic social network. This network represents friendships where nodes are individuals and edges are friendship relationships. The vector under each node represents the node’s features.

The versatility of graph representations manifests across numerous domains, each leveraging specific mathematical formulations. Here are some examples of graph structures in real-world applications:

Social networks: Social networks exemplify attributed undirected graphs where vertices represent individuals with associated feature vectors (e.g., demographics, interests), and edges capture symmetric relationships such as friendships [99, 75, 118, 94, 110, 109]. As shown in Figure 1.1, each node maintains its feature representation

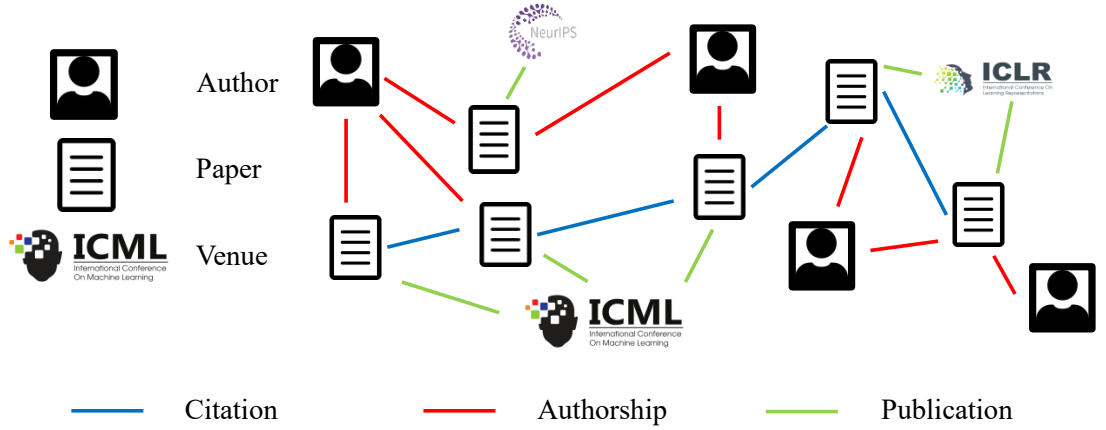


Figure 1.2: Heterogeneous Citation Network. The graph shows a citation network with three types of nodes: papers, authors, and venues. Different edge types represent various relationships: citation connections (blue lines), author-paper connections (red lines), and field-paper connections (green lines).

while undirected edges preserve the mutual nature of social connections. The real-world instances of social networks are abundant, such as Facebook, Twitter, and LinkedIn, which could contain billions of nodes and edges. People often focus on tasks such as friend recommendation [108, 133], social influence maximization [2, 80, 23], community detection [38, 71, 4] and spam detection [113, 46, 1] on social networks.

Citation networks: Citation networks are a type of heterogeneous graph that models the relationships between papers, authors, and venues. The graph shows a citation network with three types of nodes: papers, authors, and venues. Different edge types represent various relationships: citation connections (blue lines), author-paper connections (red lines), and field-paper connections (green lines).

These diverse applications share the common characteristic of irregular, interconnected data that resists traditional Euclidean analysis methods. The inherent complexity and richness of such graph-structured data necessitate specialized analytical tools capable of reasoning over both local neighborhoods and global structural patterns.

1.1.2 Graph Neural Networks

In recent years, Graph Neural Networks (GNNs) have emerged as a dominant paradigm for learning representations from graph-structured data [138, 162, 66]. GNNs extend the principles of deep learning to non-Euclidean graph domains, typically by iteratively aggregating information from neighboring nodes to learn powerful node representations through message passing frameworks [42, 168]. The foundational work includes Graph Convolutional Networks (GCNs) [66], which simplify spectral convolutions for efficient learning, Graph Attention Networks (GATs) [121], which introduce attention mechanisms for adaptive neighbor weighting, and GraphSAGE [49], which enables inductive learning on previously unseen nodes. These representations can then be utilized for various downstream tasks, such as node classification, link prediction, and graph classification, achieving state-of-the-art performance in numerous applications [119, 49, 61, 163].

The evolution of GNNs has been driven by both theoretical advances and practical needs. Early spectral approaches [9, 26] provided mathematical foundations, while spatial methods [91, 119] offered more intuitive and flexible frameworks. Recent developments include addressing over-smoothing issues [78, 15, 85], enhancing expressive power [140, 93, 103, 64, 154], and scaling to large graphs [123, 89, 84, 107].

Despite their remarkable success, the journey of GNNs towards broader and more seamless real-world applicability is still fraught with significant challenges. While GNN models have laid a strong groundwork, their practical deployment often encounters obstacles related to data availability, the complexity of graph structures, and the efficiency of the models themselves. Addressing these limitations is crucial for unlocking the full potential of GNNs and making them truly applicable across a wider spectrum of practical problems. This thesis is dedicated to exploring and overcoming some of these critical hurdles.

1.2 Problem Statement and Core Challenges

The pursuit of more applicable GNNs motivates us to confront several key challenges that currently hinder their widespread adoption and optimal performance in diverse settings. Rather than treating these as isolated issues, we organize them into three interconnected problem areas that systematically address the fundamental limitations of current GNN approaches.

1.2.1 Data Scarcity in Few-Labeled Graph Learning

Many pioneering GNN models rely heavily on the availability of abundant labeled data for effective training. However, in numerous real-world applications, obtaining high-quality labeled graph data is an expensive, time-consuming, or sometimes infeasible process: domain experts are required for accurate labeling, the cost of annotation scales with graph size, and privacy concerns often limit data sharing in sensitive domains such as healthcare and finance.

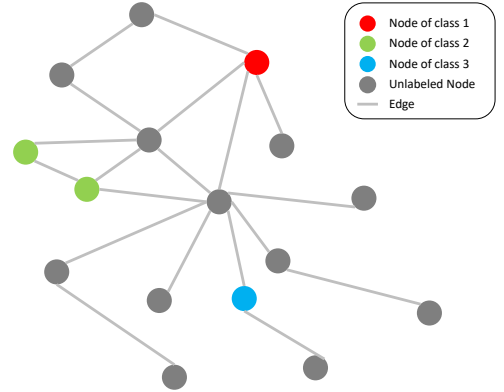


Figure 1.3: Challenge of few-labeled graph learning with only 1-2 labeled nodes per class (colored nodes).

The scarcity of labels significantly limits the performance and generalizability of GNNs in few-labeled or semi-supervised learning scenarios [85, 67] as shown in Figure 1.3, which is demonstrated in various applications including drug discovery [132, 134] and financial fraud detection [24]. Under extreme conditions where only one labeled node per class is available, existing GNNs exhibit severe performance degradation and training instability. Our empirical analysis reveals that this instability

manifests as high variance in prediction distributions across different training runs, making GNN models unreliable for practical deployment.

Traditional approaches to address this challenge include self-training methods [78, 112] that leverage high-confidence predictions as pseudo-labels, and co-training [78] techniques that combine multiple learning paradigms. However, these approaches suffer from critical limitations: (1) **Training instability** due to the random sampling of limited labeled nodes, which heavily influences the message passing process depending on graph topology; (2) **Imbalanced pseudo-label generation** that fails to maintain class distribution balance; and (3) **Poor pseudo-label quality** arising from low-confidence predictions being incorrectly included in training.

The core technical challenge lies in developing stable training procedures that can effectively propagate limited supervision signals across graph structures while maintaining prediction reliability. This requires innovations in both pseudo-labeling strategies and regularization techniques that can address the fundamental instability arising from sparse supervision.

1.2.2 Structural Complexity in Heterogeneous Graph Modeling

Real-world graphs are often intricate and multifaceted, exhibiting heterogeneous structures that encompass multiple types of nodes and edges. A particularly challenging and prevalent type is the heterogeneous graph, which comprises diverse entity types connected through various relationship types [32, 143] as shown in Figure 1.2. Such graphs encapsulate rich, diverse information and relationships that are often oversimplified or inadequately captured by GNNs designed for homogeneous graphs [88]. The applications of heterogeneous graphs span various domains, including recommendation systems [34, 19, 11] and drug discovery [44, 152].

The fundamental challenge in heterogeneous graph modeling stems from the **semantic mixing problem**. In existing approaches, when a node aggregates messages from neighbors of different types, these methods force transformations from different type-specific feature spaces into a single unified space. This forced transformation entangles semantics from different node types, leading to information loss and reduced model expressiveness.

Existing heterogeneous GNN approaches can be categorized into several paradigms, each with distinct limitations: (1) **Meta-path based methods** [130, 41] that rely on predefined or learned path patterns but suffer from high computational costs and limited applicability to graphs with many edge types; (2) **Relation-specific transformations** [104, 169] that create separate parameters for each relation type but may not effectively capture cross-type interactions; and (3) **Transformer-based architectures** [57] that apply attention mechanisms but still struggle with semantic mixing and computational complexity [160].

The core technical innovation required is a message passing mechanism that can preserve type-specific semantics throughout the learning process. This necessitates developing architectures that maintain separate semantic spaces for different node types while enabling effective information propagation across the heterogeneous graph structure.

1.2.3 Model Efficiency and Deployment Scalability

While deeper and more complex GNN architectures often yield superior performance, they also introduce significant computational overhead that creates barriers to practical deployment. The message passing framework, which forms the core of GNN effectiveness, requires iterative aggregation over node neighborhoods, leading to computational complexity that scales with both graph size and model depth. These factors render sophisticated GNN models impractical for deployment scenarios with

strict constraints.

The efficiency challenge manifests across multiple dimensions: (1) **Computational overhead** from expensive message passing operations that require accessing node neighborhoods and performing aggregation computations; (2) **Memory requirements** that grow with graph size and model complexity, limiting applicability to large-scale graphs; (3) **Inference latency** that makes real-time applications infeasible; and (4) **Hardware constraints** in edge computing environments where computational resources are severely limited. Applications requiring efficient GNN inference include recommendation systems [13], and high energy physics [58, 100].

Current approaches to address efficiency challenges include knowledge distillation methods that transfer GNN knowledge to simpler models. Existing solutions such as GLNN [159] distill GNN teachers into MLP students using soft labels, while NoSMoG [116] incorporates graph structure and robustness considerations. However, these approaches suffer from fundamental limitations: they treat both teacher GNN and student MLP as monolithic entities, focusing primarily on output-level distillation while overlooking the intrinsic knowledge embedded within GNN layer operations.

The key insight for addressing this challenge lies in recognizing that GNN layers contain two distinct types of operations: feature transformation (FT) and graph propagation (GP). FT operations share mathematical formulations with fully-connected layers in MLPs, suggesting a direct pathway for knowledge transfer. Meanwhile, GP operations possess unique smoothing properties that can be characterized and transferred through specialized distillation techniques.

1.2.4 Summary of Challenges

These three problems limit the practical applicability of Graph Neural Networks (GNNs) by data scarcity, structural complexity, and computational efficiency. They relate conceptually in that they each address a critical bottleneck in GNN deployment

and performance in real-world scenarios. Together, they form a comprehensive framework for enhancing GNNs’ robustness, expressiveness, and efficiency. The unifying principle that connects them is that they are all fundamental challenges for GNNs in real world applications. Addressing these challenges is essential for advancing GNNs towards broader practical applicability.

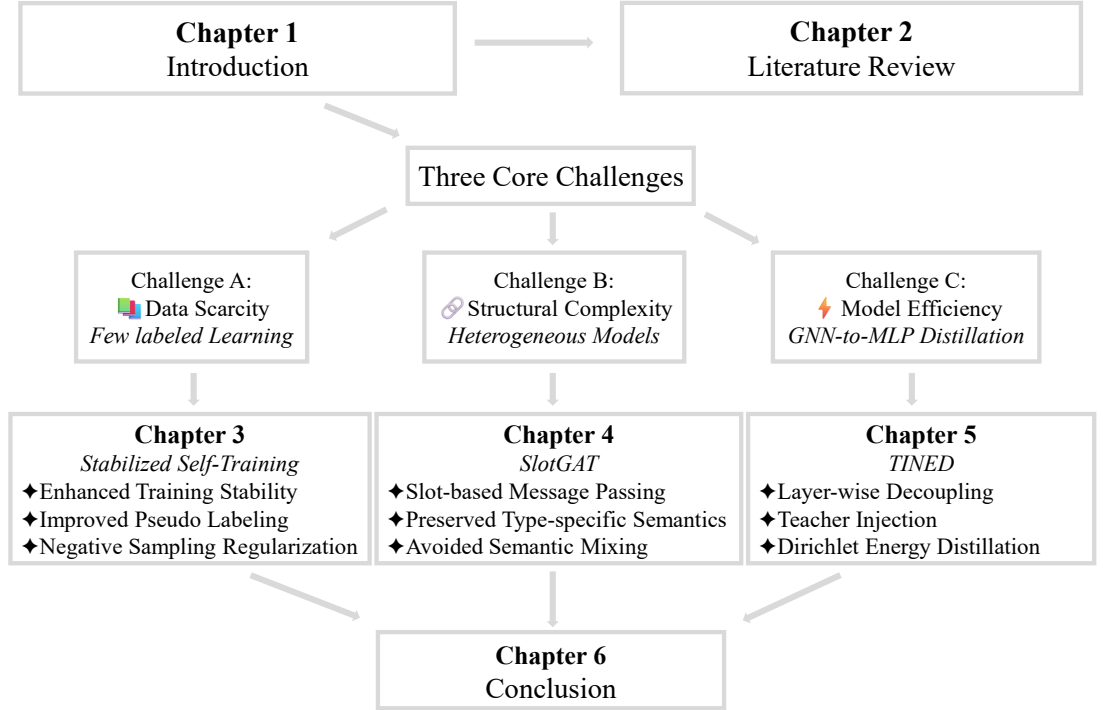


Figure 1.4: Overview of the framework of this thesis. We start with the basic concepts of graph neural networks and then introduce the three main challenges in GNN applicability and our proposed solutions: (A) Data scarcity challenge and the Stabilized Self-Training framework, (B) Heterogeneous graph complexity challenge and the SlotGAT architecture, and (C) Model efficiency challenge and the TINED distillation method. The framework is designed to address the challenges of data scarcity, structural complexity, and model efficiency in graph neural networks, and to make GNNs more applicable, robust, and efficient for a wider range of real-world challenges.

1.3 Research Questions and Objectives

Driven by the aforementioned challenges, this thesis investigates three core research questions (RQs), each paired with a corresponding declarative objective:

1. **RQ1 (Data Scarcity):** How can we enhance the effectiveness and stability of GNNs when faced with a scarcity of labeled data?

Objective: We aim to develop robust stabilized self-training frameworks to mitigate performance degradation and training instability in few-labeled graph learning contexts.

2. **RQ2 (Heterogeneity):** How can we design expressive message passing mechanisms that adeptly integrate information from diverse node and edge types?

Objective: We aim to design novel slot-based architectures that effectively capture heterogeneous structures and resolve the semantic mixing problem.

3. **RQ3 (Efficiency):** How can we compress complex GNN models into lightweight alternatives (e.g., MLPs) to facilitate easier deployment?

Objective: We aim to formulate advanced knowledge distillation methodologies that transfer GNN capabilities to computationally efficient models without sacrificing performance.

The primary goal of this research is to provide tangible solutions to these questions, contributing to the development of more applicable, robust, and efficient graph neural networks.

1.4 Main Contributions

This thesis makes several key contributions towards addressing the challenges of data scarcity, structural complexity, and model efficiency in graph neural networks:

- **Effective Stabilized Self-Training for Few-Labeled Graph Data:** To tackle the challenge of data scarcity, we propose an effective stabilized self-training framework that addresses the instability issues prevalent in existing self-training approaches [78, 112]. This approach is designed to enhance the performance of GNNs when only a limited amount of labeled data is available by leveraging unlabeled data in a robust and stable manner through stabilized pseudo-labeling and negative sampling regularization. Our work, detailed in Chapter 3, demonstrates significant improvements in learning on few-labeled graph datasets [167].
- **SlotGAT: Slot-based Message Passing for Heterogeneous Graphs:** Addressing the complexities of heterogeneous graphs, we introduce SlotGAT, a novel slot-based message passing mechanism that overcomes the semantic mixing problem inherent in existing heterogeneous GNN approaches [130, 156, 57]. SlotGAT is specifically designed to handle the diverse types of nodes and edges in heterogeneous graphs more effectively, enabling richer and more nuanced information aggregation while preserving type-specific semantics. This contribution, presented in Chapter 4, shows superior performance in learning representations for heterogeneous graph data [166].
- **TINED: GNNs-to-MLPs by Teacher Injection and Dirichlet Energy Distillation:** To improve model efficiency and facilitate deployment, we present TINED (Teacher Injection and Dirichlet Energy Distillation), which advances beyond existing knowledge distillation methods [159, 116] by directly leveraging the intrinsic knowledge embedded in GNN layers. This method distills knowledge from a powerful GNN (teacher) into a simpler MLP (student) model through novel teacher injection techniques and Dirichlet energy-based distillation that captures the smoothing properties of GNN operations [102]. By incorporating teacher injection and leveraging Dirichlet energy for distillation, TINED achieves significant model compression while maintaining competitive

performance. This work, discussed in Chapter 5, offers a practical pathway to deploying GNN-level intelligence in more lightweight forms [165].

Collectively, these contributions aim to push the boundaries of GNN applicability, making them more versatile and practical for a wider range of real-world challenges while maintaining strong theoretical foundations.

1.5 Thesis Outline

The remainder of this thesis is organized as follows:

- **Chapter 2** provides a comprehensive review of the fundamental concepts of graph neural networks, discusses existing approaches related to learning on graphs with limited supervision, modeling heterogeneous graphs, and techniques for model inferencing acceleration and knowledge distillation in the context of GNNs. It covers both classical methods [66, 121, 48] and recent advances [85, 88, 159, 116], setting the stage by identifying gaps in current research that this thesis aims to address.
- **Chapter 3** details our proposed methodology for improving GNN performance in data-scarce scenarios. This chapter corresponds to our work published in Information Sciences (INS) 2023 [167].
- **Chapter 4** presents our novel SlotGAT framework for enhanced representation learning on heterogeneous graphs. This chapter is based on our research published in the International Conference on Machine Learning (ICML) 2023 [166].
- **Chapter 5** describes our TINED approach for distilling GNNs into MLPs. This chapter reflects our work to appear in the International Conference on Machine Learning (ICML) 2025 [165].

- **Chapter 6** summarizes the main findings and contributions of this thesis. It also discusses the limitations of the current work and suggests potential directions for future research in the domain of applicable graph neural networks,

Chapter 2

Literature Review

2.1 Introduction to the Chapter

This chapter provides a comprehensive review of the existing literature pertinent to Graph Neural Networks (GNNs) and the key challenges addressed in this thesis: enhancing their applicability by tackling data scarcity, managing structural complexity in graphs, and improving model efficiency for practical deployment. We begin by laying down the foundational concepts of GNNs, followed by a focused review of current methodologies and their limitations in the areas of learning with few labels, processing heterogeneous graphs, and GNN model distillation. This review will establish the context and underscore the research gaps that motivate the novel contributions presented in the subsequent chapters.

2.2 Fundamentals of Graph Neural Networks

Graphs are powerful data structures for modeling entities and their relationships across a multitude of domains. This mathematical framework provides an elegant

means to model complex relational structures, transforming irregular, non-Euclidean data into structured mathematical objects amenable to computational analysis. The versatility of graph representations manifests extensively across numerous domains, each leveraging specific structural properties to capture domain-specific semantics.

In **social networks**, graphs naturally model symmetric relationships between individuals, where vertices represent users with associated demographic and interest features, and edges capture mutual connections such as friendships. Major platforms like Facebook, Twitter, and LinkedIn demonstrate the scale and complexity of these networks, often containing billions of nodes and edges. Critical applications include friend recommendation [108, 133], social influence maximization [2, 80, 23], community detection [38, 71, 4], and spam detection [113, 46, 1] [99, 75, 118, 94, 110, 109].

Web link networks demonstrate directed weighted graphs where asymmetric relationships carry varying importance. Here, vertices represent web pages, directed edges model hyperlinks, and weights often correspond to authority measures such as PageRank scores, capturing the hierarchical structure of web connectivity [82, 45].

Recommendation systems utilize heterogeneous graphs to model complex multi-relational data, integrating multiple entity types—users, items, categories, and contextual information—connected through various relationship types. These systems effectively capture the diverse semantics present in user-item interactions, content relationships, and collaborative filtering patterns [35, 83, 129].

In **biological and chemical networks**, graphs naturally capture multi-level structural complexity that exists both within individual molecules and across their interactions. At the molecular level, chemical compounds and biological macromolecules possess intrinsic graph structures: atoms correspond to vertices with chemical properties as features, while chemical bonds form edges with varying types (single, double, triple) and strengths. Similarly, protein structures exhibit internal graph topology where amino acid residues serve as nodes connected by various interaction types in-

cluding hydrogen bonds, disulfide bridges, and spatial proximities [39, 70].

Beyond these intrinsic molecular structures, biological systems exhibit higher-order network organizations through intermolecular interactions. Protein-protein interaction networks reveal functional relationships and regulatory pathways, drug-target interaction networks model therapeutic mechanisms, and metabolic networks capture biochemical reaction cascades [87, 61]. This multi-scale graph representation—from atomic bonds within molecules to functional associations between biomolecules—provides rich structural information that enables comprehensive computational analysis for drug discovery, protein function prediction, and systems biology understanding. The dual-level graph structures in biological domains present unique opportunities for hierarchical modeling approaches that can simultaneously capture both local molecular properties and global network effects.

These diverse applications share the common characteristic of irregular, interconnected data that resists traditional Euclidean analysis methods. The inherent complexity and richness of such graph-structured data necessitates specialized machine learning techniques capable of reasoning over both local neighborhoods and global structural patterns, while preserving the relational properties that make graph representations uniquely powerful.

Graph Neural Networks (GNNs) have emerged as the leading paradigm for machine learning on graph-structured data [138, 162, 66]. The core idea behind most GNNs revolves around a message passing or neighborhood aggregation scheme: nodes iteratively update their representations (embeddings) by aggregating feature information from their local neighborhood. This process allows GNNs to learn rich, context-aware representations that capture both node features and the underlying graph topology with the help of backpropagation.

Several typical GNN models have paved the way for subsequent research:

- **Graph Convolutional Networks (GCNs)** [66] simplified spectral graph con-

volution, providing an efficient and effective way to learn node representations by averaging neighbor features. The GCN layer can be formulated as:

$$H^{(l+1)} = \sigma \left(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)} \right) \quad (2.1)$$

where $\tilde{A} = A + I$ is the adjacency matrix with added self-connections, $\tilde{D} = \text{diag}(\tilde{A}\mathbf{1})$ is the corresponding degree matrix, $H^{(l)}$ represents node features at layer l , $W^{(l)}$ is the learnable weight matrix, and σ is an activation function. GCNs typically consist of a few layers due to challenges like over-smoothing in deeper architectures [78, 15].

- **GraphSAGE (Sample and Aggregate)** [49] introduced an inductive learning framework, enabling GNNs to generalize to unseen nodes. The GraphSAGE update rule is:

$$h_{N(v)}^{(l)} = \text{AGGREGATE}_l \left(\{h_u^{(l-1)} : u \in N(v)\} \right) \quad (2.2)$$

$$h_v^{(l)} = \sigma \left(W^{(l)} \cdot \text{CONCAT} \left(h_v^{(l-1)}, h_{N(v)}^{(l)} \right) \right) \quad (2.3)$$

where $N(v)$ denotes the neighborhood of node v , and AGGREGATE can be various functions (e.g., mean, LSTM, pooling) to combine neighborhood information.

- **Graph Attention Networks (GATs)** [121] incorporated attention mechanisms, allowing nodes to assign different importance weights to their neighbors during aggregation. The GAT layer computes:

$$e_{ij} = a(W h_i, W h_j) \quad (2.4)$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in N_i} \exp(e_{ik})} \quad (2.5)$$

$$h'_i = \sigma \left(\sum_{j \in N_i} \alpha_{ij} W h_j \right) \quad (2.6)$$

where $a(\cdot)$ is the attention function, α_{ij} represents the attention coefficient between nodes i and j , thereby learning more nuanced representations.

- **Approximated Personalized Propagation of Neural Predictions (APPNP)**

[68] applied personalized PageRank to propagate node representations through the graph. The APPNP formulation is:

$$H^{(0)} = \text{MLP}(X) \tag{2.7}$$

$$H^{(k+1)} = (1 - \alpha)\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(k)} + \alpha H^{(0)} \tag{2.8}$$

where α is the teleport probability, X are the input node features, and the iteration continues until convergence. This method could extend the GNNs to deeper architectures by mitigating over-smoothing.

These learned representations are then used for various downstream tasks, including node classification, link prediction, and graph classification, where GNNs have consistently demonstrated state-of-the-art performance [121, 49].

The selection of baseline competitors in this thesis is primarily guided by established benchmarks in the literature [67, 85, 68, 78, 112, 130, 88, 159, 116] and their direct relevance to the specific research challenges investigated. Given that our contributions specifically target the optimization of the message-passing paradigm, whether to mitigate label scarcity, handle heterogeneity, or enhance efficiency, we prioritize foundational models that are representative of this scheme and have achieved widespread adoption within the research community.

Furthermore, regarding the focus on node classification in large-scale networks, recent empirical studies [86] have demonstrated that classic GNN models (such as GCN, GAT, and GraphSAGE) can match or even exceed the performance of newer architectures like Graph Transformers when subjected to fair hyperparameter tuning. Consequently, we employ these classic GNNs as robust and comparable baselines to ensure a rigorous and focused comparative analysis of our proposed methods.

2.3 Learning on Graphs with Few Labels

The efficacy of many GNN models is often predicated on the availability of a substantial amount of labeled data for training. However, in numerous real-world scenarios, acquiring such labels is costly and time-consuming, leading to situations with very few labeled nodes. This data scarcity severely hampers GNN performance [85].

Research to mitigate this issue primarily explores two avenues:

- **Deeper GNN Architectures:** This approach explores multi-hop graph topological features to propagate training labels over the input graph. While standard GNNs like GCN [67] and APPNP [68] leverage both labeled and unlabeled nodes, they suffer from over-smoothing issues when applying many layers. Deeper architectures like DAGNN [85] address this by decoupling representation transformation and propagation, utilizing an adaptive adjustment mechanism to balance information from local and global neighborhoods. However, these methods are not specifically designed to overcome extreme label scarcity and may still suffer from performance degradation when initial labels are extremely few.
- **Self-Training on Graphs:** Self-training aims to augment the limited labeled data by iteratively generating pseudo-labels for unlabeled nodes based on the model’s own high-confidence predictions. Works like [78] and [112] have adapted self-training for GNNs. However, these methods can suffer from issues such as the generation of low-quality or imbalanced pseudo-labels, which can introduce noise and lead to unstable training, especially when initial labels are extremely few (as empirically shown in [167], Chapter 3 of this thesis).

The instability issues and problems with low-quality, imbalanced pseudo-labels in existing self-training GNN approaches highlight the need for more robust and stabilized methods for learning effectively from few-labeled graph data. This thesis aims to

address this gap by proposing a stabilized self-training framework.

2.4 Graph Neural Networks for Heterogeneous Graphs

Real-world graphs often exhibit significant structural complexity. A prominent example is the heterogeneous graph, defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \phi, \psi)$, where nodes $\mathcal{V}$ and edges $\mathcal{E}$ can have different types, managed by mapping functions ϕ and ψ respectively [56, 32, 143]. Such graphs, common in academic networks, protein interactions, and recommendation systems [35, 39], contain rich semantics tied to these types. Applying conventional (homogeneous) GNNs to heterogeneous graphs by ignoring type information can lead to suboptimal performance as valuable semantic distinctions are lost [88].

Consequently, Heterogeneous Graph Neural Networks (HGNNs) have been developed:

- **Meta-path based HGNNs:** Meta-paths are predefined sequences of node and edge types that guide neighborhood aggregation, capturing higher-order semantic relationships. HAN [130] and MAGNN [41] are notable examples that utilize hierarchical attention or sophisticated encoders along meta-paths. However, defining effective meta-paths often requires domain expertise, and automatic learning of meta-paths, as in GTN [153], can be computationally expensive and may not cover all relevant semantics.
- **Meta-path free / Relation-aware HGNNs:** These methods directly address heterogeneity without relying on explicit meta-paths. RGCN [104] uses relation-specific transformations. HetGNN [156] employs random walks and RNNs for type-specific neighbor sampling. HGT [57] leverages a Transformer architecture with type-dependent parameters. Other approaches like HetSANN [53] and SimpleHGN [88] also propose specific mechanisms for handling heterogeneity. A recent work, Space4HGNN [160], provides a unified design space for

evaluating various HGNN components.

Despite these advancements, a common challenge in existing HGNNs is the issue of *semantic mixing*. When aggregating messages from neighbors of different types, their representations are often transformed into a common feature space corresponding to the target node’s type. This forced transformation can entangle and dilute type-specific semantics (as illustrated in [166], Chapter 4 of this thesis). This observation motivates the development of HGNNs that can better preserve and distinguish semantics from different node types during message passing.

2.5 Enhancing Model Efficiency and Deployment

While sophisticated GNN architectures achieve impressive performance, their computational demands, characterized by numerous parameters and intensive message passing operations, can be prohibitive for latency-sensitive applications or deployment on resource-constrained devices [157, 155, 60]. This has spurred research into GNN model compression and acceleration.

Knowledge Distillation (KD) has emerged as a popular technique, where a smaller, faster *student* model is trained to mimic the behavior of a larger, pre-trained *teacher* GNN. KD in GNNs can be broadly categorized:

- **GNN-to-GNN Distillation:** Here, the student is also a GNN, but with fewer layers or parameters. Examples include LSP [148], FreeKD [36], and TinyGNN [141]. While these methods reduce model size, the student GNNs still rely on message passing over the graph structure, which can be a bottleneck for inference speed.
- **GNN-to-MLP Distillation:** To achieve even faster inference by eliminating message passing, recent efforts focus on distilling GNNs into Multilayer Percep-

trons (MLPs). GLNN [159] trains an MLP student using the GNN teacher’s soft label predictions. NOSMOG [116] incorporates graph structural information and adversarial robustness. Other methods like [147, 55, 161, 135, 136] also explore this direction. A typical GNN layer involves Feature Transformation (FT) and Graph Propagation (GP) operations [168].

However, many existing GNN-to-MLP distillation methods primarily focus on the final output (soft labels) of the teacher GNN, potentially overlooking the rich, fine-grained knowledge embedded within the teacher’s intermediate layers and its constituent operations (FT and GP). This suggests an opportunity to develop more effective distillation strategies that leverage this internal knowledge for better student MLP performance, a direction pursued in this thesis ([165], Chapter 5).

2.6 Summary and Research Gaps

This review has highlighted the significant progress made in Graph Neural Networks, alongside persistent challenges that limit their broader applicability. We identified three key areas:

- **Data Scarcity:** While self-training methods exist for few-labeled graph learning, they often suffer from instability and issues with pseudo-label quality, necessitating more robust frameworks.
- **Structural Complexity (Heterogeneous Graphs):** Existing HGNNs, whether meta-path based or not, can struggle with semantic mixing, where type-specific information is lost during message passing. This calls for mechanisms that better preserve distinct semantics.
- **Model Efficiency:** Distilling GNNs to MLPs is a promising avenue for fast inference, but current approaches may not fully exploit the internal, layer-wise

knowledge of the teacher GNN, indicating a need for more nuanced distillation techniques.

This thesis aims to address these identified research gaps. Subsequent chapters will introduce novel methodologies: a stabilized self-training framework to enhance learning from scarce labels (Chapter 3), a slot-based message passing mechanism for heterogeneous graphs to mitigate semantic mixing (Chapter 4), and a teacher injection and Dirichlet energy-based distillation approach for effectively compressing GNNs into MLPs (Chapter 5). These contributions collectively strive towards making GNNs more applicable, robust, and efficient for real-world tasks.

Chapter 3

Effective Stabilized Self-Training on Few-Labeled Graph Data

3.1 Introduction

A graph models the relationships between objects as the edges between nodes. Graphs are ubiquitous with a wide range of real-world applications, *e.g.*, social network analysis [99, 75], traffic prediction [82], protein interface prediction [39], and recommendation systems [35, 83]. An important task to support these applications is node classification that aims to classify the nodes in a graph into various classes. However, effective node classification is challenging, especially due to the lack of sufficient labeled data that are expensive to obtain.

To mitigate the issue, semi-supervised node classification on graphs has attracted much attention [85, 68]. It leverages a small amount of labeled nodes as well as the unlabeled nodes in a graph to train an accurate classification model. There exists a collection of Graph Neural Networks (GNNs) for semi-supervised node classification [85, 68, 67, 49]. For instance, Graph convolution networks (GCNs) rely on a message passing scheme via graph convolution operations to aggregate the neighbor-

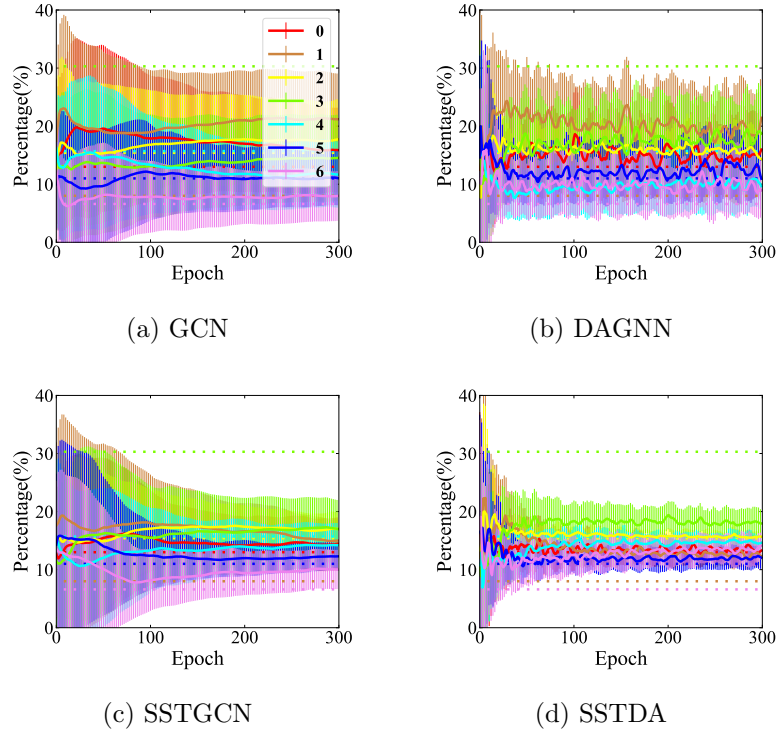


Figure 3.1: The distribution of predicted labels in different classes in Cora-1.

hood information of a node to generate node representation, which can then be used in downstream classification tasks. Despite the great success of GCNs, under the extreme cases when very few labels are given (*e.g.*, only 1 labeled node per class), the shallow GCN architecture, typically with two layers, cannot effectively propagate the training labels over the input graph, leading to inferior performance as shown in the experiments (Tables 3.3, 3.4 and 3.5). Recently, several studies try to improve classification accuracy by designing deeper GNN architectures, *e.g.*, DAGNN [85]. However, deep GNNs are still not directly designed to tackle the scarcity of labeled data.

After conducting an in-depth study, we have an interesting finding that existing GNNs suffer from an issue of *unstable* training, when labeled nodes are few. In particular, on a Cora dataset with 7 classes (see Section 3.5 for dataset details), for each run, we randomly select 1 labeled node per class as the training data (denoted as Cora-1).

Then on Cora-1, we run GCN and DAGNN 100 times with 300 epochs per run, and get the average number of predicted labels per class with standard deviation at every epoch in percentage. The statistical results of GCN and DAGNN are shown in Figures 3.1(a) and 3.1(b) respectively, where x -axis is the epoch from 0 to 300, and y -axis is the percentage of a class in the predicted node labels. There are 7 colored curves representing the average percentage of the predicted labels of the respective classes. The dashed straight lines are the ground-truth percentage of every class in the Cora dataset. The shaded areas in color represent the standard deviation. Observe that in Figure 3.1(a), GCN has high variance at different runs when predicting node labels, and the variance keeps large at late epochs, *e.g.*, 300, which indicates that GCN is quite unstable at different runs with 1 training label per class sampled randomly, leading to inferior classification accuracy as illustrated in our experiments. As shown in Figure 3.1(b), DAGNN also suffers from the issue of unstable training. The variance of DAGNN is relatively smaller than that of GCN, which provides a hint why DAGNN performs better than GCN. Nevertheless, both GCN and DAGNN yield unstable training process with large variance. Since there is only 1 labeled node per class for training, the randomly sampled training nodes can heavily influence the message passing process in GCN and DAGNN, depending on the connectivity of the training nodes to their neighborhoods over the graph topology, which result in the unstable training process observed above. In literature, there exists a collection of self-training techniques that enhance the training data by using predicted labels as *pseudo labels* for training [78, 112]. However, as identified in our empirical analysis in Section 3.4.1, these methods generate imbalanced pseudo labels, which could affect the performance.

To address the above issues when few labeled data are available, we propose a Stabilized Self-Training (SST) framework, which is readily applicable to existing GNNs to improve classification accuracy via stabilized training process. We first conduct thorough empirical and theoretical analysis to identify and explain the issues of existing methods when trained with few labeled nodes for classification. Motivated by

the analysis, in the proposed SST framework, we select a set of nodes with predicted labels of high confidence as high-quality pseudo labels, and add such pseudo labels into training data to enhance the training of next epoch, while filtering out the low-confidence predicted labels. To tackle the unstable training issue of existing GNNs, we develop a stabilized pseudo labeling technique to balance the importance of different classes in training. We then design a negative sampling regularization technique over pseudo labels to further improve node classification accuracy. In experiments, we apply our SST framework to GCN and DAGNN to get methods SSTGCN and SSTDA respectively. Figures 3.1(c) and 3.1(d) report the average percentage and standard variance of the predicted labels per class per epoch of SSTGCN and SSTDA on Cora-1 respectively. Compared with Figure 3.1(a) of GCN, obviously, the variance of SSTGCN in Figure 3.1(c) decreases quickly and becomes stable as epoch increases. SSTDA is also more stable than DAGNN, as shown in Figures 3.1(d) and 3.1(b) respectively. We conduct extensive experiments on 5 benchmarking datasets, and compare with 10 existing solutions, to evaluate the performance of the proposed SST framework. Experimental results demonstrate that SST is able to significantly improve classification accuracy of existing GNNs when only few labels are available. For instance, with the proposed SST framework, SSTGCN achieves 62.5% node classification accuracy on Cora-1, significantly improving GCN (44.6%) by 17.9%, and SSTDA obtains 66.4% accuracy on Cora-1 and outperforms DAGNN (59.8%) by a substantial margin of 6.6%.

In summary, our main contributions are as follows:

- We conduct thorough empirical and theoretical analysis, and make important findings that existing GNNs are unstable when training with few-labeled graph data, and existing self-training techniques suffer from low-quality and imbalanced pseudo labels. Our theoretical analysis provides solid explanations for the findings.

- We present a Stabilized Self-Training (SST) framework that can significantly improve classification performance by stabilizing the training process of GNNs. Based on our analysis, SST consists of a stabilized pseudo labeling technique and a negative sampling regularization technique over pseudo labels.
- We apply SST to popular GNNs, and conduct extensive experiments on 5 datasets to compare against 10 existing methods. The experimental results demonstrate the superior performance of our proposed techniques.

The rest of the paper is organized as follows. We review the related work in Section 3.2. In Section 3.3, we present the problem formulation of semi-supervised node classification on few-labeled graph data. We present our analysis and method in Section 3.4. In particular, we present the empirical analysis in Section 3.4.1, conduct theoretical analysis in Section 3.4.2, and then develop the SST framework in Section 3.4.3. Extensive experiments are reported in Section 3.5. Finally, we conclude the paper in Section 3.6. All proofs are in 3.7.

3.2 Related Work

In literature, there are two directions to address the scarcity of labeled data for semi-supervised node classification: (i) explore multi-hop graph topological features to propagate the training labels over the input graph [67, 85]; (ii) enhance the labeled data by self-training and pseudo labeling [78, 72]. Note that these two directions are not mutually exclusive, and they can be applied together on few-labeled graph data. Here we review the existing studies that are most relevant to this paper.

There exist a large collection of GNNs [85, 68, 67, 119, 91, 20, 140, 40]. We introduce the details of GCN [67] and DAGNN [85] here. GCN learns the representation of each node by iteratively aggregating the representations of its neighbors. Specifically, GCN consists of k layers, each with the same propagation rule defined as follows. At

the ℓ -th layer, the hidden representations $\mathbf{H}^{(\ell-1)}$ of previous layer are aggregated to get $\mathbf{H}^{(\ell)}$,

$$\mathbf{H}^{(\ell)} = \sigma(\hat{\mathbf{A}}\mathbf{H}^{(\ell-1)}\mathbf{W}^{(\ell)}), \ell = 1, 2, \dots, k. \quad (3.1)$$

$\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$ is the graph laplacian, where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the adjacency matrix of $\mathcal{G}$ after adding self-loops ($\mathbf{I}$ is the identity matrix) and $\tilde{\mathbf{D}}$ is a diagonal matrix with $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$. $\mathbf{W}^{(\ell)}$ is a trainable weight matrix of the ℓ -th layer, and σ is a nonlinear activation function. Initially, $\mathbf{H}^{(0)} = \mathbf{X}$, where $\mathbf{X}$ is the input feature matrix, describing the node features of all nodes in the input graph. Note that GCN usually achieves superior performance with 1 layer or 2 layers. When applying many layers to explore large receptive fields, GCN yields degraded performance, due to the over-smoothing issue [78, 73, 15]. DAGNN addresses the over-smoothing issue by decoupling representation transformation and propagation in GNNs [85]. Then it utilizes an adaptive adjustment mechanism to balance the information from local and global neighborhoods of every node. Specifically, the mathematical expression of DAGNN is as follows. DAGNN uses a learnable parameter $\mathbf{s} \in \mathbb{R}^{c \times 1}$ to adjust the weight of embeddings at different propagation level (from 1 to k).

$$\begin{aligned} \mathbf{Z} &= \text{MLP}(\mathbf{X}) \in \mathbb{R}^{n \times c} \\ \mathbf{H}_\ell &= \hat{\mathbf{A}}^\ell \cdot \mathbf{Z} \in \mathbb{R}^{n \times c}, \ell = 1, 2, \dots, k \\ \mathbf{S}_\ell &= \mathbf{H}_\ell \cdot \mathbf{s} \in \mathbb{R}^{n \times 1}, \ell = 1, 2, \dots, k \\ \hat{\mathbf{S}}_\ell &= [\mathbf{S}_\ell, \mathbf{S}_\ell, \dots, \mathbf{S}_\ell] \in \mathbb{R}^{n \times c}, \ell = 1, 2, \dots, k \\ \mathbf{X}_{out} &= \text{softmax}\left(\sum_{\ell=1}^k \mathbf{H}_\ell \odot \hat{\mathbf{S}}_\ell\right), \end{aligned}$$

where $\hat{\mathbf{A}}^\ell$ is the ℓ -th power of matrix $\hat{\mathbf{A}}$, $\odot$ is the Hadamard product, $\cdot$ is dot product, MLP is the Multilayer Perceptron and softmax operation is on the second dimension.

Apart from GCN and DAGNN, initial GNN studies apply convolution operation in the spectral domain, where the eigenvectors of the graph Laplacian are considered as the Fourier basis [51, 26]. Then GAT [119] assigns different weights to nodes in the

same neighborhood via attention mechanisms. [91] define convolutions directly in the spatial domain using mixture model CNNs. APPNP [68] adopts a propagation rule based on personalized PageRank [8], so as to gather both local and global information on graphs. GpLCN [40] utilizes the manifold structure information by p-Laplacian matrix to extract abundant features for classification. As evaluated in experiments, existing methods yield inferior performance [85, 68, 67, 119], since they are not directly designed to tackle the scarcity of labeled data.

As previously mentioned, another way to address the situation of limited labeled data is to add pseudo labels to training dataset by self-training [78]. Self-training is a general methodology [105] used in various applications. [164] suggest that selecting informative unlabeled data using a guided search algorithm can significantly improve performance over standard self-training framework. [10] mainly consider self-training for diffusion-based techniques. Recently, self-training has been adopted for semi-supervised tasks on graphs. For instance, [78] propose self-training and co-training techniques for GCN. This self-training work selects the top- k confident predicted labels as pseudo labels. Co-training technique co-trains a GCN with a random walk model to handle few-labeled data [78, 22]. [112] utilize a DeepCluster technique to refine the selected pseudo labels. Compared with existing self-training methods, our framework is different. In particular, our framework has a different strategy to select pseudo labels and also has a stabilizer to address the deficiencies of existing GNNs. Moreover, we propose a negative sampling regularization technique to further boost accuracy. Besides, in existing work, if a node is selected as a pseudo label, it will never be moved out even if the pseudo label becomes obviously wrong in later epochs. On the other hand, in our framework, we update pseudo labels in each epoch to avoid such an issue. A recent work [79] further takes node informativeness into account for pseudo-label selection.

Further, there are extensive studies on network embedding in recent years, which aims to learn a low-dimensional embedding vector per node in an unsupervised man-

ner [96, 114, 6, 120]. The learned embedding vectors can be then used in downstream tasks, including node classification. [96] use truncated random walks to learn latent representations, with the assumption that nodes are similar if they are close by random walks. [114] propose to preserve and concatenate the first-order and second-order proximity representations between nodes. G2G [6] embeds each node as a Gaussian distribution according to a ranking similarity based on the shortest path distances between nodes. DGI [120] is an embedding method based on GCNs with unsupervised objective that is to maximize mutual information between patch representations and corresponding high-level summaries of graphs. However, these unsupervised methods do not leverage labeled data, and thus are not as accurate as our methods in experiments.

While domain adaptation methods [5, 127, 62] have achieved success in CV and NLP fields by leveraging auxiliary source domains, our framework focuses on semi-supervised node classification within a single graph domain. This approach avoids dependencies on external data and circumvents the challenges associated with transferring topological information across disparate graph structures, such as molecular graphs versus social networks. We acknowledge that domain adaptation could potentially offer advantages when abundant labeled data from a highly related source domain is available, or in handling heterophilous graphs where traditional self-training assumptions of homophily may be less effective. However, our single-domain focus ensures broad applicability without requiring suitable auxiliary datasets.

3.3 Preliminaries

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ be a graph consisting of a node set $\mathcal{V}$ with cardinality n , a set of edges $\mathcal{E}$ of size m , and a feature matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, where d is the number of features. An edge in $\mathcal{E}$ connects two nodes in $\mathcal{V}$. Every node $v_i \in \mathcal{V}$ has a feature vector $\mathbf{X}_i$ that is the i -th row of $\mathbf{X}$. Let scalar c be the number of classes, and $\mathcal{C}$ be the set of class

labels ($c = |\mathcal{C}|$). We use $\mathcal{L}$ to denote the set of labeled nodes, and obviously $\mathcal{L} \subseteq \mathcal{V}$. Let $\mathcal{U}$ be the set of unlabeled nodes and $\mathcal{U} = \mathcal{V} \setminus \mathcal{L}$. Each labeled node $v_i \in \mathcal{L}$ has a one-hot vector $\mathbf{Y}_i \in \{0, 1\}^c$, indicating the class label of v_i . Under the *few-labeled setting*, $|\mathcal{L}| \ll |\mathcal{U}|$. The definition of the semi-supervised node classification problem is as follows.

Definition 1. *Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, a set of labeled nodes $\mathcal{L} \subseteq \mathcal{V}$, and a ground-truth class label $\mathbf{Y}_i \in \{0, 1\}^c$ per node $v_i \in \mathcal{L}$, assuming that each node belongs to exactly one class, Semi-Supervised Node Classification predicts the labels of the unlabeled nodes.*

In particular, the aim is to leverage a graph $\mathcal{G}$ with the labeled nodes in $\mathcal{L}$ to train a classification model/function $f(\mathcal{G}, \theta)$. The model f with trainable parameters θ outputs a matrix $\mathbf{F} \in \mathbb{R}^{n \times c}$. The i -th row $\mathbf{F}_i \in [0, 1]^c$ represents the probability vector of node $v_i \in \mathcal{V}$. We adopt the popular cross-entropy loss to train the model. Given a node v_i , its loss $\mathbf{L}(\mathbf{Y}_i, \mathbf{F}_i)$ of $\mathbf{F}_i$ with respect to its true class label $\mathbf{Y}_i$ is

$$\mathbf{L}(\mathbf{Y}_i, \mathbf{F}_i) = - \sum_{j=1}^c \mathbf{Y}_{i,j} \ln(\mathbf{F}_{i,j}),$$

where $\mathbf{Y}_{i,j}$ is the j -th value in $\mathbf{Y}_i$ and $\mathbf{F}_{i,j}$ is the j -th value in $\mathbf{F}_i$.

During the training process, at a certain epoch, given the matrix $\mathbf{F}$, let $\tilde{\mathbf{Y}}_{i,j}$ satisfying Eq. (3.2) be the predicted label of node v_i at the epoch. In particular, if $\mathbf{F}_{i,j}$ is the *largest* element in vector $\mathbf{F}_i$, $\tilde{\mathbf{Y}}_{i,j}$ is 1, otherwise, 0. We say that, with *confidence* $\mathbf{F}_{i,j}$, node v_i has class label $\mathcal{C}_j$.

$$\tilde{\mathbf{Y}}_{i,j} = \begin{cases} 1 & \text{if } j = \arg \max_{j'} \mathbf{F}_{i,j'}, \\ 0 & \text{otherwise,} \end{cases} \quad (3.2)$$

3.4 Our Method and Analysis

We first conduct empirical analysis in Section 3.4.1 to identify the issues of existing techniques, and then perform theoretical analysis in Section 3.4.2 to reveal the latent reasons of the empirical findings. Lastly, we present our technical designs of Stabilized Self-Training (SST) in Section 3.4.3.

3.4.1 Empirical Analysis

We conduct empirical analysis to reveal that existing techniques suffer from imbalanced and low-confidence pseudo labels during the training stage. We first verify that during the training process, nodes with higher confidence tend to be predicted more accurately. On the Cora dataset with 7 classes, each with 1 labeled node for training, Figure 3.2a exhibits the *positive* correlation between the confidence and the accuracy scores of GCN at the 100-th training epoch with confidence interval 0.1. In particular, for each confidence interval (x -axis), we report the percentage of unlabeled nodes with predicted labels matching their ground-truth labels (*i.e.*, accuracy in y -axis). Obviously, as the confidence becomes higher, the accuracy increases. On the other hand, if a method generates predictions with low confidence, its performance tends to be inferior.

Then in Figure 3.3, we provide empirical evidence that most nodes are with low confidence, especially at the early epochs of the training, which will hamper the performance. Specifically, in Figures 3.3a, 3.3b, and 3.3c, we illustrate the distributions of the confidence scores of all nodes in unlabeled set $\mathcal{U}$ on Cora, at early (20-th), middle (100-th), and late (500-th) epochs, respectively. Observe that at the early epoch in Figure 3.3a, 95.4% of the unlabeled nodes are with very low confidence (< 0.2). If pseudo labels are generated based on the low confidences at early epochs, then such pseudo labels are inaccurate as verified above. Subsequently, they will

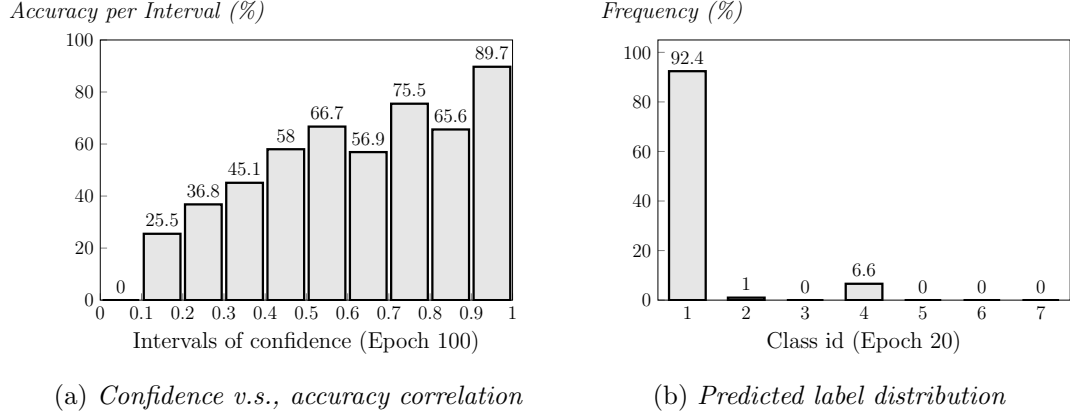


Figure 3.2: Training GCN on Cora with 1 labeled node per class (Cora-1)

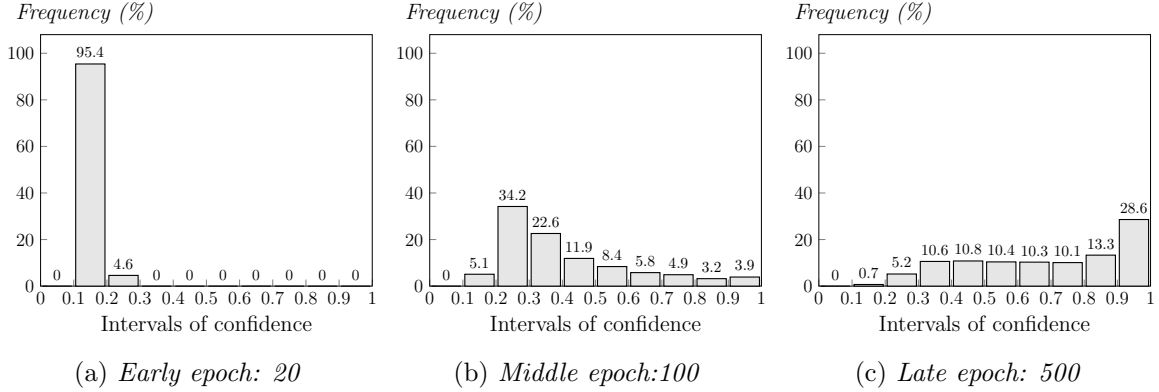


Figure 3.3: Confidence distributions of GCN Training on Cora-1.

severely influence the training of later epochs, resulting to unstable training process and unexpected performance degradation. Further, low-confidence nodes are still the majority during middle epochs, *e.g.*, 100-th epoch in Figure 3.3b. The number of low-confidence nodes decreases significantly at very late epochs, *e.g.*, 500-th epoch in Figure 3.3c. These observations indicate that the low-confidence nodes keep exposing significant influence during the major part of the whole training process.

Further, Figure 3.2b shows the distribution of the predicted labels of the nodes in $\mathcal{U}$ at the 20-th epoch on Cora in a single run. Obviously, the distribution of the labels is highly imbalanced, *i.e.*, 92.4% of the nodes are with class 1 and 6.6% are with

class 4. The imbalanced class labels (*i.e.*, class 1 in Figure 3.2b) will heavily affect the direction of gradient descent during the training process, imposing the model to learn weights resulting to imbalanced predictions, which is often not the ground-truth case. If too many low-confidence nodes are wrongly assigned to a single class at early epochs, existing GNNs, such as GCN, which rely on propagation and transformation over graph topology, will suffer from unstable performance as reported in Figure 3.1.

3.4.2 Theoretical Analysis

The semi-supervised node classification problem studied in this paper naturally fits the *transductive learning* setting, since all labeled and unlabeled data in the input graph $\mathcal{G}$ are known, and no more new data will be added [33]. In the following, we analyze the problem from the perspective of *gradient descent*, to theoretically explain the findings made in Section 3.4.1.

Under the *ideal case* where we have the labels of *all* nodes in $\mathcal{V}$ (the population), given a classifier $f(\mathcal{G}, \theta)$ with output probability vectors $\mathbf{F}_i$ for all nodes v_i in $\mathcal{V}$, the population loss L_{pop} is computed as follows. Ideally, the objective is to minimize L_{pop} by evaluating *population gradient* $\nabla_{\theta} L_{pop}$, and find optimal parameters θ^* .

$$L_{pop} = \mathbb{E}_{v_i \sim \mathcal{U}(\mathcal{V})} \mathbf{L}(\mathbf{Y}_i, \mathbf{F}_i),$$

where $\mathcal{U}(\mathcal{V})$ is the uniform distribution over node set $\mathcal{V}$.

However, in practice, due to the scarcity of labeled data (*i.e.*, $|\mathcal{L}| \ll |\mathcal{V}|$), it is impossible to directly evaluate $\nabla_{\theta} L_{pop}$. Therefore, as explained in Section 3.2, self-training method considers both the labeled nodes and the predicted labels of unlabeled nodes (as pseudo labels) for training. In particular, during a training epoch, the predicted labels of unlabeled nodes in Eq. 3.2 are regarded as the pseudo labels of the nodes. Then a *pseudo loss* L_{pse} is used to approximate the population loss L_{pop} . Also, *pseudo gradient* $\nabla_{\theta} L_{pse}$ is used to approximate $\nabla_{\theta} L_{pop}$, in order to minimize

the loss [98, 92]. Usually the self-training loss L_{pse} and its gradient $\nabla_{\theta} L_{pse}$ can be written as

$$L_{pse} = \mathbb{E}_{v_i \sim U(\mathcal{L})} \mathbf{L}(\mathbf{Y}_i, \mathbf{F}_i) + \lambda \cdot \mathbb{E}_{v_i \sim U(\mathcal{U})} \mathbf{L}(\tilde{\mathbf{Y}}_i, \mathbf{F}_i), \quad (3.3)$$

$$\nabla_{\theta} L_{pse} = \mathbb{E}_{v_i \sim U(\mathcal{L})} \nabla_{\theta} \mathbf{L}(\mathbf{Y}_i, \mathbf{F}_i) + \lambda \cdot \mathbb{E}_{v_i \sim U(\mathcal{U})} \nabla_{\theta} \mathbf{L}(\tilde{\mathbf{Y}}_i, \mathbf{F}_i), \quad (3.4)$$

where $U(\mathcal{L})$ and $U(\mathcal{U})$ are the uniform distributions over labeled node set $\mathcal{L}$ and unlabeled node set $\mathcal{U}$, respectively.

We rewrite the population gradient $\nabla_{\theta} L_{pop}$ in a similar format by breaking $\mathcal{V}$ into $\mathcal{L}$ and $\mathcal{U}$,

$$\begin{aligned} \nabla_{\theta} L_{pop} &= \mathbb{E}_{v_i \sim U(\mathcal{V})} \nabla_{\theta} L(\mathbf{Y}_i, \mathbf{F}_i) \\ &= \frac{|\mathcal{L}|}{|\mathcal{V}|} \mathbb{E}_{v_i \sim U(\mathcal{L})} \nabla_{\theta} \mathbf{L}(\mathbf{Y}_i, \mathbf{F}_i) + \frac{|\mathcal{U}|}{|\mathcal{V}|} \mathbb{E}_{v_i \sim U(\mathcal{U})} \nabla_{\theta} L(\mathbf{Y}_i, \mathbf{F}_i). \end{aligned}$$

Then we derive a bound on the difference between $\nabla_{\theta} L_{pop}$ and $\nabla_{\theta} L_{pse}$ in Eq. 3.5 with proof in 3.7.1. Specifically, let $\lambda = \frac{|\mathcal{U}|}{|\mathcal{L}|}$, and assume that any gradient satisfies a bounded norm (*i.e.*, $\|\nabla_{\theta} \mathbf{L}\| \leq \Theta$, for any loss $\mathbf{L}$), which is a common assumption [170]. Then the difference between $\nabla_{\theta} L_{pop}$ and $\nabla_{\theta} L_{pse}$ is bounded by

$$\begin{aligned} & \left\| \nabla_{\theta} L_{pse} - \frac{|\mathcal{V}|}{|\mathcal{L}|} \nabla_{\theta} L_{pop} \right\| \\ &= \frac{|\mathcal{U}|}{|\mathcal{L}|} \cdot \left\| \mathbb{E}_{v_i \sim U(\mathcal{U})} \left[\nabla_{\theta} \mathbf{L}(\tilde{\mathbf{Y}}_i, \mathbf{F}_i) - \nabla_{\theta} \mathbf{L}(\mathbf{Y}_i, \mathbf{F}_i) \right] \right\| \\ &\leq \frac{|\mathcal{U}| \cdot \mathbf{P}_{v_i \sim U(\mathcal{U})}(\tilde{\mathbf{Y}}_i \neq \mathbf{Y}_i)}{|\mathcal{L}|} \cdot \mathbb{E}_{v_i \sim U(\mathcal{U})} \left[\left\| \nabla_{\theta} \mathbf{L}(\tilde{\mathbf{Y}}_i, \mathbf{F}_i) - \nabla_{\theta} L(\mathbf{Y}_i, \mathbf{F}_i) \right\| \mathbf{1}_{\tilde{\mathbf{Y}}_i \neq \mathbf{Y}_i} \right] \\ &\leq \frac{2\Theta|\mathcal{U}|}{|\mathcal{L}|} \cdot \mathbf{P}_{v_i \sim U(\mathcal{U})}(\tilde{\mathbf{Y}}_i \neq \mathbf{Y}_i), \end{aligned} \quad (3.5)$$

where $\mathbf{P}_{v_i \sim U(\mathcal{U})}(\tilde{\mathbf{Y}}_i \neq \mathbf{Y}_i)$ is the probability that a randomly sampled node $v_i \in \mathcal{U}$ has a wrongly predicted label.

Obviously, $\mathbf{P}_{v_i \sim U(\mathcal{U})}(\tilde{\mathbf{Y}}_i \neq \mathbf{Y}_i)$ is the classification error on unlabeled set $\mathcal{U}$. Observe that the bound at the last line in Eq. 3.5 mainly relies on the quality of pseudo labels

$\tilde{\mathbf{Y}}_i$ of nodes v_i in $\mathcal{U}$. In other words, if we have low-quality pseudo labels from $\mathcal{U}$, then $\mathbf{P}_{v_i \sim \mathcal{U}}(\tilde{\mathbf{Y}}_i \neq \mathbf{Y}_i)$ tends to be large, leading to a large difference between $\nabla_{\theta} L_{pop}$ and $\nabla_{\theta} L_{pse}$ and consequently resulting to sub-optimal performance. This situation is likely to happen at early epochs when most nodes have low confidence but are still selected as pseudo labels for training as illustrated in Figure 3.3a of Section 3.4.1.

Moreover, at the early epochs when many nodes are with low confidence, the imbalanced predicted labels (*e.g.*, Figure 3.2b) may make $\mathbf{P}_{v_i \sim \mathcal{U}}(\tilde{\mathbf{Y}}_i \neq \mathbf{Y}_i)$ even larger. In the following, we prove that if we use imbalanced predicted labels as pseudo labels, this will lead to inferior performance. To facilitate the analysis, we define that the distribution of ground-truth labels is ρ -balanced as follows. Specifically, for any two distinct classes $a, b \in \mathcal{C}$, the balance ratio between them is upper bounded by ρ , if and only if

$$\max_{a, b \in \mathcal{C}} \frac{\mathbf{P}_{v_i \sim \mathcal{U}}(\mathbf{Y}_i = a)}{\mathbf{P}_{v_i \sim \mathcal{U}}(\mathbf{Y}_i = b)} \leq \rho. \quad (3.6)$$

Let $c' = \arg \max_{c \in \mathcal{C}} \mathbf{P}_{v_i \sim \mathcal{U}}(\tilde{\mathbf{Y}}_i = c)$ be the class with the max predicted probability. We define that the predicted labels are η -imbalanced if and only if

$$\max_{b \in \mathcal{C}, b \neq c'} \frac{\mathbf{P}_{v_i \sim \mathcal{U}}(\tilde{\mathbf{Y}}_i = c')}{\mathbf{P}_{v_i \sim \mathcal{U}}(\tilde{\mathbf{Y}}_i = b)} \geq \eta. \quad (3.7)$$

Then we present Lemma 1 that provides a lower bound of the classification error $\mathbf{P}_{v_i \sim \mathcal{U}}(\tilde{\mathbf{Y}}_i \neq \mathbf{Y}_i)$.

Lemma 1. *If the ground-truth label distribution is ρ -balanced and the predicted label distribution is η -imbalanced, we can get the lower bound of the classification error as follows,*

$$\mathbf{P}_{v_i \sim \mathcal{U}}(\tilde{\mathbf{Y}}_i \neq \mathbf{Y}_i) \geq \frac{\eta}{\eta + |\mathcal{C}| - 1} - \frac{\rho}{\rho + |\mathcal{C}| - 1}. \quad (3.8)$$

Proof. The proof of Lemma 1 is in 3.7.2. □

Lemma 1 indicates that if the predicted labels are highly imbalanced compared with ground-truth labels (*i.e.*, η is large while ρ is small), the classification error is large.

In practice, the imbalance ratio η can be arbitrarily large, particularly at the early training epochs. For instance, as shown in Figure 3.2b, the imbalance ratio η is up to 14 when using GCN on Cora. However, the ground-truth balance ratio ρ of Cora is just 4.8. Based on Lemma 1, the large discrepancy between η and ρ indicates a large classification error $\mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\tilde{\mathbf{Y}}_i \neq \mathbf{Y}_i)$, as demonstrated in Figure 3.1(a) when training with GCN. Further, considering Eq. 3.5, this will result in an inaccurate approximation of gradient $\nabla_{\theta} L_{pse}$ and lead to ineffective and unstable gradient descent process to train the classification model.

3.4.3 Stabilized Self-Training (SST) Framework

Motivated by the analysis above, in what follows, we develop the SST framework that not only augments training data with high-quality pseudo labels but also stabilizes the training process, in order to achieve superior performance.

Stabilized Pseudo Labeling

We first explain how to choose pseudo labels and then introduce the loss function of stabilized pseudo labeling. In particular, in a training epoch, for every unlabeled node $v_i \in \mathcal{U}$, we first get N_i , the number of nodes with the same predicted label as v_i ,

$$N_i = \left| \left\{ v_j \in \mathcal{U} \mid \tilde{\mathbf{Y}}_i = \tilde{\mathbf{Y}}_j \right\} \right| \quad (3.9)$$

As shown in Figure 3.1, in existing GCN and DAGNN, the distribution of the predicted labels is unstable with large variance during the training process. We also observe that during the early epochs, most predicted labels are with low confidence (Figure 3.3), low confidence indicates low accuracy (Figure 3.2a), and the predicted labels are imbalanced (Figure 3.2b). If we select too many low-confidence predicted labels as the pseudo labels of the same class, this will hamper the training process

and result in inferior performance. To reduce such unstable situation, we propose to use N_i as a stabilizer for pseudo labeling. In particular, in our loss function, we assign weights inversely proportional to N_i , to mitigate the imbalance issue. If a class is with a larger N_i , its importance will be lower, and vice versa. For an unlabeled node $v_i \in \mathcal{U}$, its predicted label may have low confidence score. We do not want to add such low-confidence labels into the training of subsequent epochs. Instead, we only choose those unlabeled nodes with high-confidence predicted labels as pseudo labels to be augmented into the training data. In particular, an unlabeled node v_i is selected to be a node with *pseudo label* in next epoch, if its confidence is beyond a threshold β , as shown below. We use $\mathcal{U}'$ to denote all unlabeled nodes selected with pseudo labels,

$$\mathcal{U}' = \left\{ v_i \in \mathcal{U} \mid \max_j \mathbf{F}_{i,j} > \beta \right\} \quad (3.10)$$

where $\beta \in [0, 1]$ is a threshold controlling the extent of cautious selection for self-training. A larger threshold means stricter selection of pseudo labels.

Then we present the loss of the stabilized pseudo labeling technique in Eq. (3.11). Specifically, we design $\frac{1}{N_i+1}$ as the stabilizer of the training process, to overcome the deficiencies of existing GNNs illustrated in Figure 3.1. The intuition is that, if an unlabeled node v_i is predicted to be in a class with many pseudo labels, its importance in the loss function should be reduced. In other words, our stabilized self-training loss reduces the impact of classes with many pseudo labeled nodes, which is especially useful to rectify the training process when the predictions in the early epochs are incorrect or less confident, compared with ground truth. Further, L_{sp} only considers high-confident nodes in $\mathcal{U}'$ defined in Eq. 3.10, and filter out low-confidence nodes in the training epochs.

$$L_{sp} = \sum_{\forall v_i \in \mathcal{U}'} \frac{1}{N_i + 1} \cdot \mathbf{L}(\tilde{\mathbf{Y}}_i, \mathbf{F}_i) \quad (3.11)$$

Compared with existing techniques [78], our stabilized pseudo labeling technique has major differences. First, we develop the stabilizer to re-weight the importance of

pseudo labels in the loss function, so as to address the unstable issue of existing GNNs. Second, we select only those nodes with high-confidence pseudo labels satisfying β threshold, and adaptively update the pseudo labels *per epoch*, meaning that a pseudo label in previous epoch will be removed in the next epoch if its confidence becomes low. On the other hand, existing methods keep a pseudo label once it is selected and never remove it in later epochs, which may harm the training quality if the pseudo label is wrong compared with ground truth.

Negative Sampling Regularization

Under extreme cases when labeled nodes are very few (*e.g.*, 1 labeled node per class), we further design a negative sampling regularization technique to improve performance. Existing studies mainly use negative sampling to get embeddings in an unsupervised manner [120, 149, 90]. On the contrary, we customize negative sampling for the semi-supervised node classification task, and apply it over *labels* instead of embeddings. Intuitively, the label of a node v should be distant to the label of another node u if these two nodes are faraway on the input graph $\mathcal{G}$. Specifically, a positive sample is a node v_i in $\mathcal{L}$ or $\mathcal{U}'$. We sample a set $\mathcal{I}$ of positive samples from $\mathcal{L} \cup \mathcal{U}'$ uniformly at random. The negative samples of a positive sample v_i are the nodes that are not directly connected to v_i in graph $\mathcal{G}$. For *each* positive sample v_i in $\mathcal{I}$, we sample a fixed-size set $\mathcal{J}_i$ of negative samples uniformly at random. For a positive-negative pair (v_i, v_j) , compared with the $\tilde{\mathbf{Y}}_i$ of $v_i \in \mathcal{L} \cup \mathcal{U}'$, the intention is to let the output vector $\mathbf{F}_j$ of v_j to be as different as possible. Without ambiguity, here we use the symbol $\tilde{\mathbf{Y}}_i$ to represent the pseudo label for node v_i in $\mathcal{U}'$ or ground-truth label of node v_i in $\mathcal{L}$. Denote $\mathbf{1}$ as the all-one vector in $\mathbb{R}^c$. Then the total loss of all positive-negative pairs is

$$\mathbf{L}_{neg} = \sum_{\forall v_i \in \mathcal{I}} \sum_{\forall v_j \in \mathcal{J}_i} \frac{1}{|\mathcal{I}| \cdot |\mathcal{J}_i|} L(\tilde{\mathbf{Y}}_i, \mathbf{1} - \mathbf{F}_j). \quad (3.12)$$

In Stabilized Self-Training (SST), we adopt negative-sampling regularization over

alternative strategies like entropy minimization or consistency regularization. We posit that entropy minimization is redundant, as self-training inherently encourages high-confidence (low-entropy) predictions through pseudo-label generation. Similarly, consistency regularization is implicitly achieved via the aggregation mechanisms of GNNs (e.g., GCN, APPNP), which naturally enforce local smoothness across connected nodes. In contrast, negative-sampling regularization provides a complementary perspective by exploiting the topological information of non-connected pairs. This approach, common in graph representation learning, allows the model to leverage broader structural knowledge, which is particularly critical when supervision signals are scarce.

Overall Objective Function

The overall loss $\mathbf{L}_{total}$ combines the stabilized pseudo labeling loss and the negative sampling loss in Eq. (3.11) and Eq. (3.12) by

$$\mathbf{L}_{total} = \frac{1}{|\mathcal{L}|} \cdot \sum_{\forall v_i \in \mathcal{L}} \mathbf{L}(\mathbf{Y}_i, \mathbf{F}_i) + \lambda_1 L_{sp} + \lambda_2 \mathbf{L}_{neg}, \quad (3.13)$$

where λ_1 and λ_2 are factors controlling the impact of these two losses.

Algorithm 1 shows the pseudo-code of the SST framework over GNNs, and it takes as input a graph $\mathcal{G}$ with labeled nodes $\mathcal{L}$ and unlabeled nodes $\mathcal{U}$. Note that SST can be instantiated over either a shallow or a deep GNN, *e.g.*, GCN and DAGNN introduced in Section 3.2. The output of Algorithm 1 is the learned classification model f with trainable parameters θ . At Line 3, SST initializes the trainable parameters θ by Xavier [43]. Then from Lines 4 to 13, SST trains the classification model per epoch t iteratively, until convergence or the max number T of epochs is reached. We adopt a widely-used early-stopping technique for convergence [85, 150]. Specifically, after half of the max number of epochs (half of $T = 1000$ epochs in experiments), given a tolerance duration (100 epochs), if the validation loss of the current epoch is higher than the smallest validation loss of the past 100 epochs in the tolerance duration, the

Algorithm 1 SST Framework Over GNNs

```
1: Input: Graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$  with labeled node set  $\mathcal{L}$  and unlabeled node set  $\mathcal{U}$ 
2: Output: the learned classifier  $f(\cdot, \theta)$ .
3: Generate initial parameter  $\theta$  for model  $f(\cdot, \cdot)$ .
4: for each epoch  $t = 0, 1, 2, \dots, T$  do
5:   Use GNN to compute prediction  $\mathbf{F} \leftarrow f(\mathcal{G}, \theta)$ 
6:   Get high confidence set  $\mathcal{U}'$  and its stabilizing factor  $\frac{1}{N_i+1}$  per node  $v_i$  (Section 3.4.3)
7:   Get positive samples and corresponding negative samples using  $\mathcal{L} \cup \mathcal{U}'$  and  $\mathcal{G}$  (Section 3.4.3)
8:   Get  $\mathbf{L}_{total}$  of current epoch by Eq. (3.13) (Section 3.4.3)
9:   Update model parameters by  $\theta \leftarrow \text{Adam Optimizer}(\theta, \text{gradient} = \nabla_{\theta} \mathbf{L}_{total})$ .
10:  if Convergence then
11:    break
12:  end if
13: end for
```

model converges and terminates. The model achieving the smallest validation loss is selected as final. At Line 5, SST first uses a GNN to obtain the forward prediction output $\mathbf{F}$. Then at Line 6, SST detects the pseudo-labeled set $\mathcal{U}'$ and obtains the stabilizer $\frac{1}{N_i+1}$ of each node v_i in $\mathcal{U}'$, after which, at Line 7 we perform negative sampling to obtain positive samples $\mathcal{I}$ and negative samples $\mathcal{J}_i$. At Line 8, SST computes loss $\mathbf{L}_{total}$ of current epoch according to Eq. (3.13). And at Line 9, SST updates model parameters θ for next epoch by Adam optimizer [65].

3.5 Experiments

We evaluate SST against 10 competitors for semi-supervised node classification on 5 benchmark graph datasets. All experiments are conducted on a machine powered by

Table 3.1: Dataset Statistics

Dataset	Cora	Citeseer	PubMed	Cora-full	Ogbn-arxiv
# of Nodes	2708	3327	19717	19793	169343
# of Edges	5429	4732	44338	65311	1166243
# of Features	1433	3703	500	8710	128
# of Classes	7	6	3	67	40

an Intel(R) Xeon(R) E5-2603 v4 @ 1.70GHz CPU, 131GB RAM, 16.04.1-Ubuntu, and 4 Nvidia Geforce 1080ti Cards with Cuda version 10.2. Source codes of all competitors are obtained from the respective authors. Our SST framework is implemented in Python, using libraries including PyTorch [95] and PyTorch Geometric [37].

3.5.1 Implementation Details

We instantiate our SST framework over a 2-layer GCN and a deep DAGNN to demonstrate the effectiveness and applicability of SST. The instantiation of SST over GCN and DAGNN are dubbed as SSTGCN and SSTDA respectively. SSTGCN and SSTDA have parameters (i) inherited from GCN and DAGNN and (ii) developed in SST. Hence, we first tune the best parameters of the base models under each classification task setting on each dataset and report this result for them for a fair comparison.

Base models (GCN and DAGNN).

In the base models, we tune parameters, including L_2 regularization rate with search space in $\{1e-2, 5e-3, 1e-3, 5e-4, 1e-4, 5e-5, 0\}$ and dropout rate in $\{0.5, 0.8\}$. For DAGNN, the level k of propagation after MLP is searched in $\{10, 15, 20\}$. The number of hidden units of GCN and MLP (in DAGNN) is 64 units without bias. The number of layers of GCN and MLP (in DAGNN) is 2 layers. The learning rate of Adam Optimizer is 0.01. The activation function is RELU. The maximum number

of training epochs is 1000. Moreover, early stopping is triggered when the validation loss is smaller than the average validation loss of previous 100 epochs, and the current epoch is beyond 500 epochs.

SSTGCN and SSTDA. After finding the best hyper parameters of the base models, we then tune the parameters in SST. λ_1 is searched in $\{0.1, 1\}$ and λ_2 is searched in $\{0, 0.1, 1\}$. Stabilizing enabler searches in $\{\text{True}, \text{False}\}$. The number of positive and negative samples $(|\mathcal{I}|, |\mathcal{J}_i|)$ is searched in $\{(1, 10), (2, 5), (5, 2), (10, 1)\}$. For instance, $(2, 5)$ means that we sample 2 positive nodes and then for each positive node, we sample 5 negative nodes.

3.5.2 Datasets and Competitors

Datasets. Table 3.1 shows the statistics of the 5 real-world graphs used in our experiments. We list the number of nodes, edges, features and classes in every dataset. Specifically, the 5 datasets are Cora [106], Citeseer [106], Pubmed [106], Core-full [6], and Obgn-arxiv [54], all of which are widely used in existing studies [85, 78].

Competitors. We compare with 10 existing solutions, including LP (Label Propagation) [137], G2G [6], DGI [120], GCN [67], GAT [119], APPNP [68], DAGNN [85], STs [78], LCGCN and LCGAT in [139]. In particular, GCN, GAT, APPNP, and DAGNN are GNNs. DGI and G2G are unsupervised network embedding methods. STs represents the four variants in [78], including Self-Training, Co-Training, Union, and Intersection; we summarize the best results among them as the results of STs. We use the parameters suggested in the original papers of the competitors to tune their models, and report the best results of the competitors. Notice that for unsupervised network embedding methods, including DGI and G2G, after obtaining the embeddings, we use logistic regression to train a node classifier over the embedding [120].

Table 3.2: Absolute accuracy improvements (in percentage) of SST (*i.e.*, SSTGCN and SSTDA) over base models GCN and DAGNN on 4 datasets, averaged over 100 random data splits.

# Labels per class	Cora					CiteSeer				
	1	3	5	10	20	1	3	5	10	20
GCN	44.6	63.8	71.3	77.2	81.4	40.4	53.5	61.0	65.8	69.5
SSTGCN	62.5	72.8	75.8	80.7	82.5	56.2	66.4	68.0	70.2	72.1
Improvement	+17.9	+9.0	+4.5	+3.5	+1.1	+14.2	+12.9	+7.0	+4.4	+2.6
DAGNN	59.8	72.4	76.7	80.8	83.7	46.5	58.8	63.6	67.9	71.2
SSTDA	66.4	77.6	79.8	82.2	84.1	48.5	65.9	67.9	69.8	72.1
Improvement	+6.6	+5.2	+3.1	+1.4	+0.4	+2.0	+7.1	+4.3	+1.9	+0.9

# Labels per class	PubMed					Cora-Full				
	1	3	5	10	20	1	3	5	10	20
GCN	55.5	66.0	70.4	74.6	78.7	24.5	41.4	48.1	55.8	60.2
SSTGCN	60.8	67.8	71.6	76.1	79.4	30.8	44.9	49.4	56.6	60.9
Improvement	+5.3	+1.8	+1.2	+1.5	+0.7	+6.3	+3.5	+1.3	+0.8	+0.7
DAGNN	59.4	69.5	72.0	76.8	80.1	27.3	43.2	49.8	55.8	60.4
SSTDA	61.0	72.1	74.9	78.2	80.6	27.6	44.4	51.1	56.8	61.2
Improvement	+1.6	+2.6	+2.9	+1.4	+0.5	+0.3	+1.2	+1.3	+1.0	+0.8

3.5.3 Experimental Settings

We evaluate our framework and the competitors on semi-supervised node classification tasks with various settings. In particular, for each graph dataset, we repeat experiments on 100 random data splits as suggested in [85, 78] and report the average performance. For each graph dataset, we vary the number of labeled nodes per class in $\{1, 3, 5, 10, 20\}$, where 1, 3, 5 represent the very few-labeled settings. Following convention in existing work [85], we explain what a random data split is, as follows.

Table 3.3: Accuracy results (in percentage) on PubMed and Cora-full respectively, averaged over 100 random data splits. (The **best** accuracy is in **bold**.)

# Labels per class	PubMed					Cora-full				
	1	3	5	10	20	1	3	5	10	20
SSTGCN	60.8	67.8	71.6	76.1	79.4	30.8	44.9	49.4	56.6	60.9
SSTDA	61.0	72.1	74.9	78.2	80.6	27.6	44.4	51.1	56.8	61.2
LP	55.7	61.9	63.5	65.2	66.4	26.3	32.4	35.1	38.0	41.0
G2G	55.2	64.5	67.4	72.0	74.3	25.8	36.4	43.3	49.3	54.3
DGI	55.1	63.4	65.3	71.8	73.9	26.2	37.9	46.5	55.3	59.8
STs	55.1	65.4	69.7	74.0	78.5	29.2	43.6	48.9	53.4	60.8
GCN	55.5	66.0	70.4	74.6	78.7	24.5	41.4	48.1	55.8	60.2
DAGNN	59.4	69.5	72.0	76.8	80.1	27.3	43.2	49.8	55.8	60.4
APPNP	54.8	66.9	70.8	76.0	79.4	24.3	41.5	48.5	55.3	60.1
GAT	52.7	64.4	69.4	73.7	73.5	24.8	41.0	47.5	54.7	59.9
LCGCN	56.6	69.2	72.6	74.6	80.0	26.7	43.9	49.2	55.9	60.5
LCGAT	49.5	59.2	62.3	70.2	65.3	27.4	43.2	48.4	55.0	60.1

For example, when the number of labeled nodes per class on Cora is 3 (denoted as Cora-3), since Cora has 7 classes, we randomly pick 3 nodes per class, combining together as a training set of size 21 (*i.e.*, the labeled node set $\mathcal{L}$), and then, among the remaining nodes, we randomly select 500 nodes as a validation set, and 1000 nodes as a test set. Every data split consists of a training set, a validation set, and a test set. Classification accuracy is defined as the fraction of the testing nodes with class labels correctly predicted by the learned classifier.

Table 3.4: Accuracy results (in percentage) on Cora and CiteSeer respectively, averaged over 100 random data splits. (The **best** accuracy is in **bold**.)

# Labels per class	Cora					CiteSeer				
	1	3	5	10	20	1	3	5	10	20
SSTGCN	62.5	72.8	75.8	80.7	82.5	56.2	66.4	68.0	70.2	72.1
SSTDA	66.4	77.6	79.8	82.2	84.1	48.5	65.9	67.9	69.8	72.1
LP	51.5	60.5	62.5	64.2	67.3	30.1	37.0	39.3	41.9	44.8
G2G	54.5	68.1	70.9	73.8	75.8	45.1	56.4	60.3	63.1	65.7
DGI	55.3	70.9	72.6	76.4	77.9	46.1	59.2	64.1	67.6	68.7
STs	53.1	67.3	72.5	76.2	79.8	37.2	51.8	60.7	67.4	70.2
GCN	44.6	63.8	71.3	77.2	81.4	40.4	53.5	61.0	65.8	69.5
DAGNN	59.8	72.4	76.7	80.8	83.7	46.5	58.8	63.6	67.9	71.2
APPNP	44.7	66.3	74.1	79.0	81.9	34.6	52.2	59.4	66.0	71.8
GAT	41.8	61.7	71.1	76.0	79.6	32.8	48.6	54.9	60.8	68.2
LCGCN	63.6	74.4	77.5	80.4	82.4	55.3	59.0	68.4	70.3	72.1
LCGAT	58.7	74.5	77.5	79.7	82.6	50.9	66.3	68.5	70.9	71.5

3.5.4 Overall Experimental Results

In Table 3.2, we first report the absolute improvements of our SST applied over base models GCN and DAGNN, when varying the number of labeled nodes per class in $\{1, 3, 5, 10, 20\}$. The accuracy performance of SSTGCN (resp. GCN) and SSTDA (resp. DAGNN) on each of the 4 datasets are obtained by averaging over 100 random data splits. The overall observation is that SSTGCN and SSTDA consistently achieve superior performance compared with their respective base models GCN and DAGNN, often by a significant margin, which validates the power of the proposed SST framework, to be readily applicable to boost the performance of existing GNNs. For instance, on Cora-1, SSTGCN has accuracy 62.5%, while the accuracy of GCN is

44.6%, which indicates that SST improves GCN by 17.9%. Also, on Cora-1, SSTDA improves DAGNN by 6.6%. Moreover, observe that when labels are sufficient (*e.g.*, Cora-20 and CiteSeer-20 in Table 3.2), our methods SSTGCN and SSTDA are still better than GCN and DAGNN respectively, further validating the effectiveness of the proposed SST, by stabilizing the training process as shown in Figures 3.1(c) and 3.1(d) of Section 4.1.

Table 3.3 reports the classification accuracy (in percentage) of all methods on PubMed and Cora-full when varying the number of labeled nodes per class in $\{1, 3, 5, 10, 20\}$. The first two rows are the performance of SSTGCN and SSTDA, while the remaining rows are the performance of the 10 competitors. On PubMed, SSTDA consistently achieves the highest accuracy among all methods under all settings in $\{1, 3, 5, 10, 20\}$. For instance, on PubMed-3, SSTDA has accuracy 72.1%, while the best competitor’s performance is 69.5%. Further, on Cora-full in Table 3.3, SSTGCN and SSTDA achieve the highest accuracy under the settings $\{1, 3\}$ and $\{5, 10, 20\}$ respectively (in bold), compared with all competitors. The results in Table 3.3 demonstrate the effectiveness of our SST framework to boost classification performance over existing methods. Table 3.4 reports the classification results of all methods on Cora and CiteSeer under all settings in $\{1, 3, 5, 10, 20\}$. On Cora, observe that SSTDA obtains the highest accuracy under all settings. For instance, on Cora-1, SSTDA has accuracy 66.4%, while that of the best competitor LCGCN is 63.6%. On CiteSeer in Table 3.4, SSTGCN has the best performance on CiteSeer-1 and CiteSeer-3 and CiteSeer-20, while achieving similar performance compared with LCGCN and LCGAT on CiteSeer-5 and CiteSeer-10. Summing up, the proposed SST framework is effective to boost classification accuracy as validated in Tables 3.3 and 3.4, especially under the situation when very few labeled nodes are available.

Table 3.5 reports the classification accuracy (in percentage) of all methods on Ogbn-arxiv, when varying the number of labeled nodes per class in $\{1, 3, 5, 10, 20\}$. The first two rows are the performance of SSTGCN and SSTDA, while the remaining rows

Table 3.5: Accuracy results (in percentage) on Ogbn-arxiv, averaged over 100 random data splits. (The **best** accuracy is in **bold**.)

# Labels per class	Ogbn-arxiv				
	1	3	5	10	20
SSTGCN	25.57	41.12	44.42	50.66	53.99
SSTDA	27.20	42.66	47.29	53.76	57.13
DGI	15.25	24.35	31.89	38.55	43.34
STs	22.49	34.44	40.57	49.14	52.23
GCN	25.11	36.32	40.90	48.42	53.76
DAGNN	25.18	40.72	46.66	52.57	56.07
APNP	25.21	39.74	45.58	49.51	53.50
GAT	25.86	38.52	45.58	50.83	53.53
LCGCN	25.96	38.62	44.68	52.06	57.09
LCGAT	26.93	37.31	43.09	47.72	55.05

are the performance of the competitors. SSTDA consistently achieves the highest accuracy among all methods under all settings. For instance, on Ogbn-arxiv-3, SSTDA has accuracy 42.66%, while the best competitor’s performance is 40.72%. Further, comparing with the corresponding base models, SSTGCN (resp. SSTDA) is consistently better than GCN (resp. DAGNN). The results in Table 3.5 demonstrate the effectiveness of our SST framework to improve classification performance.

Note that our SST framework is integrated into existing GNN models (GCN and DAGNN). Therefore, SSTGCN and SSTDA inherit the efficiency of their corresponding base models, with moderate extra overheads to perform the stabilized self-training techniques proposed in this paper. We report the total training time and training time per epoch of all methods on Ogbn-arxiv in Table 3.6, to show that the methods are able to scale to large graphs. Since different models converge at different epochs, their total training time varies. The total training time of all models are

Table 3.6: Total training time in seconds (s) and training time per epoch in milliseconds (ms) on Ogbn-arxiv.

# Labels per class	Total training time (s)					Training time per epoch (ms)				
	1	3	5	10	20	1	3	5	10	20
SSTGCN	87.4	90.8	105.7	92.8	107.3	158.7	171.6	219.7	180.8	205.7
SSTDA	144.1	218.7	134.7	119.3	159.1	257.1	256.7	234.6	156.6	187.4
DGI	95.0	95.0	96.8	94.1	94.3	190.7	189.9	190.4	190.6	190.4
STs	46.1	45.4	51.3	52.6	30.0	55.0	55.1	55.3	55.5	55.4
GCN	32.4	54.5	56.3	56.3	56.5	53.0	53.4	53.0	53.1	53.4
DAGNN	83.3	110.3	122.2	158.2	156.6	150.9	150.8	150.7	151.8	148.3
APPNP	61.1	71.3	107.8	120.0	119.6	113.6	113.1	113.3	113.5	113.2
GAT	93.8	94.1	94.0	93.7	94.5	89.9	90.3	90.1	89.7	90.5
LCGCN	83.5	84.1	84.6	84.9	85.0	80.61	80.5	80.6	80.7	80.9
LCGAT	96.7	97.2	97.3	97.3	97.4	93.1	93.5	93.6	93.5	93.1

affordable (within 4 minutes). For instance, on Ogbn-arxiv-10, SSTDA converges in 119.3 seconds, while DAGNN costs 158.2 seconds and APPNP needs 120.0 seconds. On Ogbn-arxiv-1, SSTGCN converges in 87.4 seconds, while GAT and LCGAT need 93.8 and 96.7 seconds respectively. In terms of training time per epoch, compared with base models (*e.g.*, DAGNN), SSTDA has similar training time per epoch with moderate extra overheads. In summary, our methods consistently achieve the highest effectiveness, while having similar efficiency and scalability performance compared with existing methods.

3.5.5 Ablation Study

We conduct ablation study to evaluate the contributions of the techniques of SST presented in Section 3.4.3. In particular, let base models be either GCN or DAGNN.

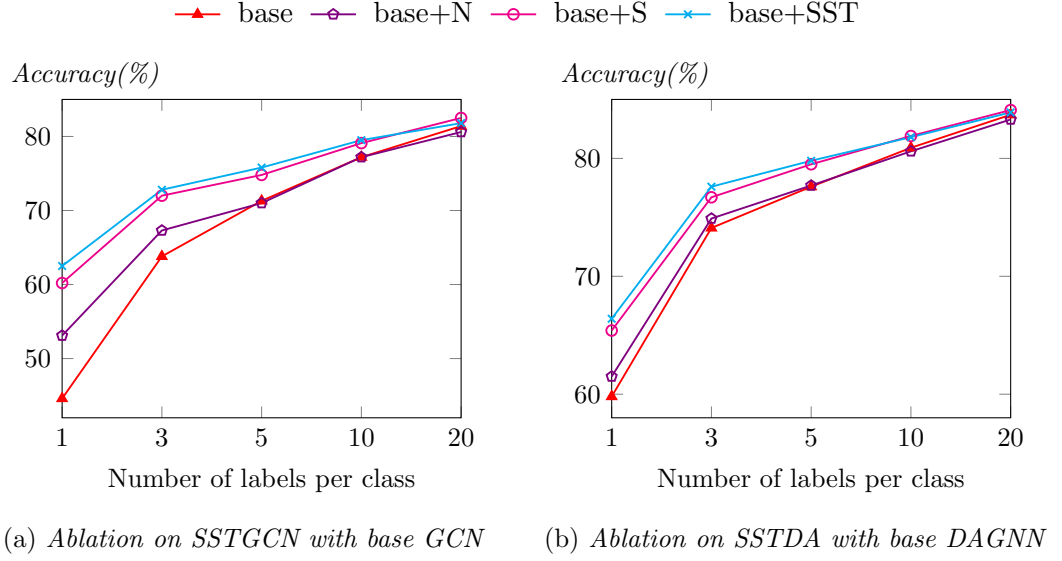
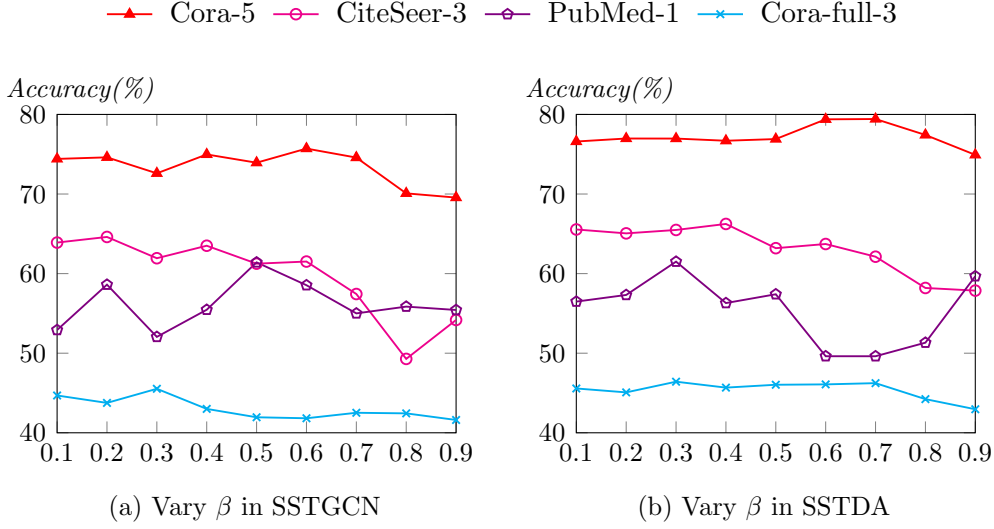


Figure 3.4: Ablation study of SST on Cora.

Then denote base+SST as the method with the whole SST framework enabled (*i.e.*, SSTGCN or SSTDA), base+S as the method with only stabilized pseudo labeling loss in Eq. (3.11) enabled, and base+N as the method with only negative sampling loss in Eq. (3.12) enabled. Figures 3.4a and 3.4b report the ablation results of SSTGCN and SSTDA respectively, on Cora when varying the number of labels per class in $\{1, 3, 5, 10, 20\}$.

Observe that the accuracy of base+S and base+N are always better than the base models, *i.e.*, GCN and DAGNN, indicating the effectiveness of the proposed pseudo labeling and negative sampling techniques in SST. Further, the accuracy of base+SST is always the highest under almost all settings. The ablation study demonstrates the power of our proposed techniques to improve classification accuracy.

We further vary β from 0.1 to 0.9, and report the performance of SSTGCN in Figure 3.5a, for node classification on Cora-5, CiteSeer-3, PubMed-1, and Cora-full-3. The corresponding number of pseudo labels when varying β in SSTGCN is reported in Table 3.7. As β increases, the number of pseudo labels decreases (Table 3.7). For SSTGCN, the best result is achieved at $\beta = 0.6, 0.2, 0.5, 0.3$ for Cora-5, CiteSeer-3,

Figure 3.5: Vary β . Results are averaged over 20 runs.Table 3.7: The number of pseudo labels when varying β in SSTGCN, averaged over 20 runs.

β	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Task									
Cora-5	2708	2707	2630	2368	2013	1714	1235	855	369
CiteSeer-3	3327	3327	3325	3307	3247	3118	2907	2436	1535
PubMed-1	19717	19717	19717	19250	16463	12349	9246	5307	1871
Cora-full-3	19789	19704	19366	18879	18006	17041	15553	13616	10233

PubMed-1, and Cora-full-3 respectively as shown in Figure 3.5a, which exhibits the trade off between the confidence and the number of pseudo labels decided by Eq. (3.10). Figure 3.5b and Table 3.8 report the accuracy of SSTDA and the number of pseudo labels respectively when varying β . The best result is achieved at $\beta = 0.6, 0.4, 0.3, 0.3$ for node classification on Cora-5, CiteSeer-3, PubMed-1, and Cora-full-3 respectively. In Table 3.8, the number of pseudo labels decreases as β increases.

Table 3.8: The number of pseudo labels when varying β in SSTDA, averaged over 20 runs.

β Task	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Cora-5	2707	2707	2675	2535	2233	1994	1678	1346	817
CiteSeer-3	3326	3327	3290	3092	2721	2305	1843	1244	575
PubMed-1	19717	19717	19717	19691	19472	18674	17733	16219	13843
Cora-full-3	19741	18892	16976	14678	12235	9914	7475	5258	2914

3.6 Conclusion

This paper presents Stabilized Self-Training (SST), an effective framework for semi-supervised node classification on few-labeled graph data. We identify that existing GNNs are with unstable performance under few-labeled settings. We also conduct extensive empirical and theoretical analysis to provide solid explanations for the observations. SST is designed with the consideration of the analysis, and achieves superior performance on graphs with extremely few labeled nodes. SST consists of a stabilized pseudo labeling technique and a negative sampling regularizer over pseudo labels. The effectiveness of SST is evaluated via extensive experiments. In the future, we plan to enhance SST by investigating other unsupervised techniques, and also implement SST on top of more GNN architectures to further demonstrate its applicability.

3.7 Proof

3.7.1 Proof of Eq. 3.5

Proof. The first inequality of Eq. 3.5 is from the property of conditional expectation. Specifically, for a random variable $X = \nabla_{\theta} \mathbf{L}(\tilde{\mathbf{Y}}_i, \mathbf{F}_i) - \nabla_{\theta} \mathbf{L}(\mathbf{Y}_i, \mathbf{F}_i)$ and a partition of sampling space into two parts:

$$A = \{v_i \sim \mathcal{U}(\mathcal{U}) | \tilde{\mathbf{Y}}_i \neq \mathbf{Y}_i\}$$

$$B = \{v_i \sim \mathcal{U}(\mathcal{U}) | \tilde{\mathbf{Y}}_i = \mathbf{Y}_i\}$$

Then the following holds since $\mathbb{E}[X|B] = 0$:

$$\begin{aligned} \mathbb{E}[X] &= \mathbb{E}[X|A]\mathbf{P}(A) + \mathbb{E}[X|B]\mathbf{P}(B) \\ &= \mathbb{E}[X|A]\mathbf{P}(A) \end{aligned}$$

Based on Jensen's inequality, the first inequality of Eq. 3.5 holds.

$$\begin{aligned} \mathbb{E}[X] &= \|\mathbb{E}[X|A]\mathbf{P}(A)\| \\ &\leq \mathbf{P}(A)\mathbb{E}[\|X\| | A]. \end{aligned}$$

The second inequality is from the bounded gradient norm assumption and the triangle property of the norm, such that

$$\begin{aligned} \left\| \nabla_{\theta} \mathbf{L}(\tilde{\mathbf{Y}}_i, \mathbf{F}_i) - \nabla_{\theta} \mathbf{L}(\mathbf{Y}_i, \mathbf{F}_i) \right\| &\leq \left\| \nabla_{\theta} \mathbf{L}(\tilde{\mathbf{Y}}_i, \mathbf{F}_i) \right\| + \left\| \nabla_{\theta} \mathbf{L}(\mathbf{Y}_i, \mathbf{F}_i) \right\| \\ &\leq 2\Theta \end{aligned}$$

Then we have

$$\begin{aligned} \frac{|\mathcal{U}| \cdot \mathbf{P}(A)}{|\mathcal{L}|} \cdot \mathbb{E}[\|X\| | B] &\leq \frac{2\Theta|\mathcal{U}|}{|\mathcal{L}|} \cdot \mathbf{P}(A) \\ &= \frac{2\Theta|\mathcal{U}|}{|\mathcal{L}|} \cdot \mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\tilde{\mathbf{Y}}_i \neq \mathbf{Y}_i) \end{aligned}$$

Thus Eq. 3.5 holds. □

3.7.2 Proof of Lemma 1

The proof of Lemma 1 needs Lemma 2 and 3 that are presented and proved as follows.

Lemma 2. *If the ground-truth distribution of labels is ρ -balanced, the maximal probability of a class will be bounded as*

$$\max_{y \in \mathcal{C}} \mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\mathbf{Y}_i = y) \leq \frac{\rho}{\rho + |\mathcal{C}| - 1}.$$

Proof of Lemma 2. One can easily derive from Eq. 3.6 that for any $y \in \mathcal{C}$, we have

$$\mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\mathbf{Y}_i = y) \geq \frac{1}{\rho} \mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\mathbf{Y}_i = y'').$$

By the property of probability, denote $y'' = \arg \max_{y \in \mathcal{C}} \mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\mathbf{Y}_i = y)$

$$\begin{aligned} 1 &= \sum_{y \in \mathcal{C}} \mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\mathbf{Y}_i = y) \\ &= \mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\mathbf{Y}_i = y'') + \sum_{y \in \mathcal{C}, y \neq y''} \mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\mathbf{Y}_i = y) \\ &\geq \mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\mathbf{Y}_i = y'') + \frac{1}{\rho} \sum_{y \in \mathcal{C}, y \neq y''} \mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\mathbf{Y}_i = y'') \\ &= (1 + \frac{|\mathcal{C}| - 1}{\rho}) \cdot \mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\mathbf{Y}_i = y''). \end{aligned}$$

Then we have

$$\max_{y \in \mathcal{C}} \mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\mathbf{Y}_i = y) \leq \frac{\rho}{\rho + |\mathcal{C}| - 1}.$$

□

Lemma 3. *If the distribution of pseudo labels is η -imbalanced, we have the lower bound of the maximal probability of a class as*

$$\max_{y \in \mathcal{C}} \mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\tilde{\mathbf{Y}}_i = y) \geq \frac{\eta}{\eta + |\mathcal{C}| - 1}.$$

Proof of lemma 3. One can easily get from Eq.3.7 that for any $y \in \mathcal{C}$,

$$\mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\tilde{\mathbf{Y}}_i = y) \leq \frac{1}{\eta} \mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\tilde{\mathbf{Y}}_i = y').$$

Due to the property of probability,

$$\begin{aligned}
 1 &= \sum_{y \in \mathcal{C}} \mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\tilde{\mathbf{Y}}_i = y) \\
 &= \mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\tilde{\mathbf{Y}}_i = y') + \sum_{y \in \mathcal{C}, y \neq y'} \mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\tilde{\mathbf{Y}}_i = y) \\
 &\leq \mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\tilde{\mathbf{Y}}_i = y') + \frac{|\mathcal{C}| - 1}{\eta} \mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\tilde{\mathbf{Y}}_i = y').
 \end{aligned}$$

Thus $\mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\tilde{\mathbf{Y}}_i = y') = \max_{y \in \mathcal{C}} \mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\tilde{\mathbf{Y}}_i = y) \geq \frac{\eta}{\eta + |\mathcal{C}| - 1}$. $\square$

After getting Lemma 2 and 3, in the following, we prove Lemma 1.

Proof of lemma 1. We expand the classification error by the pseudo labels

$$\mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\tilde{\mathbf{Y}}_i \neq \mathbf{Y}_i) = \sum_{y \in \mathcal{C}} [\mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\tilde{\mathbf{Y}}_i \neq \mathbf{Y}_i, \tilde{\mathbf{Y}}_i = y)]. \quad (3.14)$$

Let $y' = \arg \max_{y \in \mathcal{C}} \mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\tilde{\mathbf{Y}}_i = y)$, we have

$$\begin{aligned}
 \sum_{y \in \mathcal{C}} [\mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\tilde{\mathbf{Y}}_i \neq \mathbf{Y}_i, \tilde{\mathbf{Y}}_i = y)] &\geq \mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\tilde{\mathbf{Y}}_i \neq \mathbf{Y}_i, \tilde{\mathbf{Y}}_i = y') \\
 &= \sum_{y \in \mathcal{C}, y \neq y'} \mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\mathbf{Y}_i = y, \tilde{\mathbf{Y}}_i = y') \\
 &\geq \sum_{y \in \mathcal{C}, y \neq y'} [\mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\mathbf{Y}_i = y) \cdot \mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\tilde{\mathbf{Y}}_i = y')] \\
 &= [1 - \mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\mathbf{Y}_i = y')] \cdot \mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\tilde{\mathbf{Y}}_i = y').
 \end{aligned}$$

We are interested in the max and min values of $\mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\tilde{\mathbf{Y}}_i = y')$ and $\mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\mathbf{Y}_i = y')$ respectively. From Lemma 2, we have

$$\begin{aligned}
 \mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\mathbf{Y}_i = y') &\leq \max_{y \in \mathcal{C}} \mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\mathbf{Y}_i = y) \\
 &\leq \frac{\rho}{\rho + |\mathcal{C}| - 1}.
 \end{aligned}$$

From Lemma 3, we can directly derive

$$\mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\tilde{\mathbf{Y}}_i = y') \geq \frac{\eta}{\eta + |\mathcal{C}| - 1}.$$

Thus we can get

$$\begin{aligned}\mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\tilde{\mathbf{Y}}_i = y') - \mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\mathbf{Y}_i = y') &\geq \frac{\eta}{\eta + |\mathcal{C}| - 1} \left(1 - \frac{\rho}{\rho + |\mathcal{C}| - 1}\right) \\ &\geq \frac{\eta}{\eta + |\mathcal{C}| - 1} - \frac{\rho}{\rho + |\mathcal{C}| - 1}.\end{aligned}$$

Combining all above, we have the lower bound of classification error

$$\mathbf{P}_{v_i \sim \mathcal{U}(\mathcal{U})}(\tilde{\mathbf{Y}}_i \neq \mathbf{Y}_i) \geq \frac{\eta}{\eta + |\mathcal{C}| - 1} - \frac{\rho}{\rho + |\mathcal{C}| - 1}.$$

□

Chapter 4

SlotGAT: Slot-based Message Passing for Heterogeneous Graphs

4.1 Introduction

Heterogeneous graphs with node and edge types [56, 32, 143], are ubiquitous in many real applications, *e.g.*, protein prediction [39], recommendation [35, 146], social analysis [99, 75], and traffic prediction [45]. Figure 4.1(a) displays a heterogeneous academic graph with 5 nodes in 3 types, *i.e.*, author, paper, venue. Usually edge type is related to the node types of the two ends of an edge. For instance, the edge between v_1 (author) and v_2 (paper) is an authorship edge; v_2 and v_5 (paper) have a citation edge.

Heterogeneous graphs have attracted great research attentions. Conventional graph neural networks (GNNs) [49, 68, 66, 119, 91, 85] can be applied by regarding heterogeneous graphs as homogeneous ones. GNNs are good enough, but still ignore the heterogeneous semantics [88]. Hence, several heterogeneous graph neural networks (HGNNs) are developed [130, 153, 169, 156, 41, 57, 53, 104, 12, 126, 125, 129, 88, 160]. Some HGNNs rely on meta-paths, either predefined manually [130, 41] or learned au-

tomatically [153], which incur extra costs. Then there are studies of relation oriented HGNNs [169, 104], HGNNs with random walks and RNNs [156], empowering GNNs for HGNNs [88], etc. [160] define a unified design space for HGNNs.

It is known that different node types have different semantics, and naturally should have different impacts [130, 88, 57]. However, when aggregating the messages from different types of nodes, the semantics are mixed in existing work, which may hamper the effectiveness [130, 153, 169, 41]. Specifically, in existing message passing layers, when a node v aggregates messages from its neighbors u that may be in different node type, they force the representation of u in another type-specific feature space to be transformed to the feature space of v 's type, and then aggregate the transformed message to v . We argue that this forced transformation mixes different feature spaces, making representations entangled with each other. This semantic mixing issue is illustrated in Figure 4.1(c). Given the heterogeneous graph in the figure, every node is with a *single* initial representation as input. Then in Layer 1, when node v_2 (paper type id 1) aggregates the message from its neighbor v_1 (author type id 0) in a different node type, it first applies a transformation $\tau(0, 1)$ to convert the representation of v_1 from author's feature space to paper's feature space, and then aggregates it to paper v_2 's representation. Similar forced transformation also exists when aggregating v_3 (venue) to v_2 (paper) in Layer 1. Instead of converting all v 's neighboring representations to v 's feature space, if we separate and preserve the impacts of different node-type features to v , we could learn more effective representations.

Hence, we propose SlotGAT, which has separate message *slots* of different node types (*i.e.*, feature spaces), with dedicated attention mechanisms to measure the importance of slots. SlotGAT conducts *slot-based message passing processes* to alleviate the semantic mixing issue. We explain the idea of slot-based message passing in Figure 4.1(b). For a node v , we maintain a slot for every node type in the heterogeneous graph, *e.g.*, every node with 3 slots in Figure 4.1(b) for author, paper, venue types, respectively. When initializing the input slot representations of v_2 (paper), the slot

for paper type (slot 1) is initialized by v_2 's features, while other slots (slots 0, 2) of v_2 for author and venue are initialized as empty. In Layer 1, when aggregating neighbors to v_2 , the slot 0 message from v_1 is aggregated to the corresponding slot 0 of v_2 , without mixing with the representations in other slots (*i.e.*, other node types). The slot 2 message from v_3 (venue) is aggregated to the respective slot 2 of v_2 . Obviously, though v_2 is in paper type, it can maintain slot representations of other node types, compared with existing methods in Figure 4.1(c) where v_2 mixes the representations of v_1 , v_2 , v_3 , v_5 in different node types. In the layers of SlotGAT, we also design an *attention-based aggregation mechanism* that considers all slot representations of a node and its neighbors as well as edge type representations for aggregation. In downstream tasks, to effectively integrate the slot representations of a node, after the last layer of SlotGAT, we develop a *slot attention* technique to learn the importance of different slots. Our analysis indicates that SlotGAT can preserve different semantics into slots.

Compared with numerous existing methods, SlotGAT achieves superior performance on all datasets under various evaluation metrics. We also conduct model analysis, including statistical significance test, visualization, and ablation analysis, to analyze the effectiveness of SlotGAT.

We summarize our contributions as follows:

- We propose a novel slot-based heterogeneous graph neural network model SlotGAT for heterogeneous graphs.
- We develop a new slot-based message passing mechanism, so that semantics of different node-type feature spaces do not entangle with each other.
- We further design attention mechanisms to effectively process the representations in slots.
- Extensive experiments demonstrate the superiority of SlotGAT on real-world heterogeneous graphs.

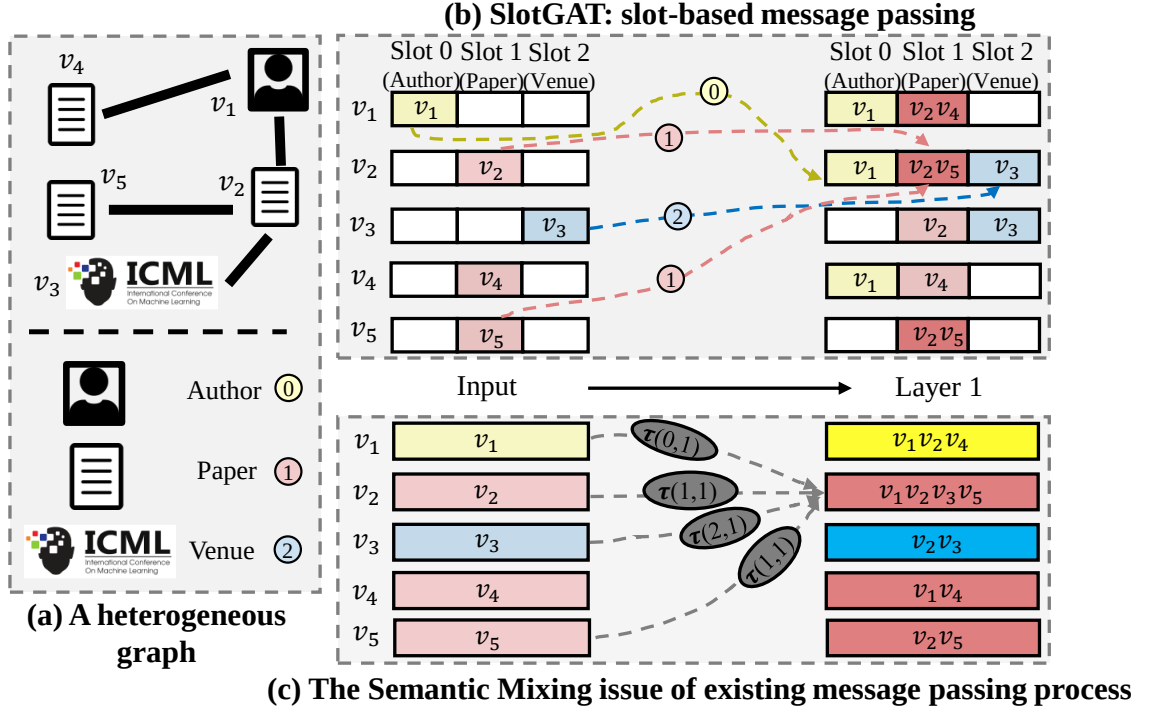


Figure 4.1: (a) A heterogeneous graph. (b) The slot-based message passing in the proposed SlotGAT: every node has 3 slots, corresponding to 3 node-type feature spaces. For example, in the input layer, v_2 's slot 1 is initialized by its features since v_2 is in type 1, while slots 0 and 2 of v_2 are empty. The message passing in SlotGAT is slot specific (colored dashed arrows), *e.g.*, neighboring slot messages passed to the slots of the same node type on node v_2 in Layer 1. (c) The semantic mixing issue: every node maintains a single representation; in Layer 1, v_2 aggregates message from v_1 by firstly applying transformation $\tau(0,1)$ to convert v_1 's representation in type 0 to the feature space of v_2 in type 1, which mixes the two feature spaces.

4.2 Preliminary

A heterogeneous graph is $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \phi, \psi)$, where $\mathcal{V}$ is the set of nodes, $\mathcal{E}$ is the set of edges, ϕ is a node type mapping function, and ψ is an edge type mapping function. Every node v has node type $\phi(v)$, and every edge $e = (v, u)$ has edge type $\psi(e)$,

also denoted as $\psi(v, u)$. Let Φ and Ψ be the set of all node types and the set of all edge types respectively in graph $\mathcal{G}$. Assume that node types in Φ are represented by consecutive integers starting from 0, and denote t as the t -th node type. Let n be the number of nodes, *i.e.*, $n = |\mathcal{V}|$, and m be the number of edges, *i.e.*, $m = |\mathcal{E}|$. For a heterogeneous graph $|\Psi| + |\Phi| > 2$.

A node v has a feature vector $\mathbf{x}_v$. For node type $t \in \Phi$, all type- t nodes $v \in \{v \in \mathcal{V} | \phi(v) = t\}$ have the same feature dimension $d_0^t = d_0^{\phi(v)}$, *i.e.*, $\mathbf{x}_v \in \mathbb{R}^{d_0^{\phi(v)}}$. Nodes of different types can have different feature dimensions [88].

4.3 Related Work

Homogeneous GNNs handle graphs without node/edge types [66, 85, 9, 26, 119, 91, 20, 68]. GCN [66] simplifies spectral networks on graphs [9] into its GNN form. GAT [119] introduces self-attention to GNNs. Then, there is a plethora of HGNNs to handle heterogeneous graphs. Meta-paths are used in [130, 41, 153]. HAN [130] consists of a hierarchical attention mechanism to capture node-level importance between nodes and semantic-level importance of meta-paths. MAGNN [41] is an enhanced method with several meta-path encoders to encode all the information along meta-paths. Both MAGNN and HAN require meta-paths generated manually. Graph transformer network (GTN) [153] can automatically learn meta-paths with graph transformation layers. For heterogeneous graphs with many edge types, meta-path based methods are not easily applicable, due to the high cost on obtaining meta-paths. There are methods treating heterogeneous graphs as relation graphs to develop graph neural network methods. RSHN is a relation structure-aware HGNN [169], which builds coarsened line graph to get edge features and adopts a Message Passing Neural Network (MPNN) [42] to propagate node and edge features. RGCN [104] splits a heterogeneous graph to multiple subgraphs by building an independent adjacency matrix for each edge type. Furthermore, HGT [57] is a graph transformer model to

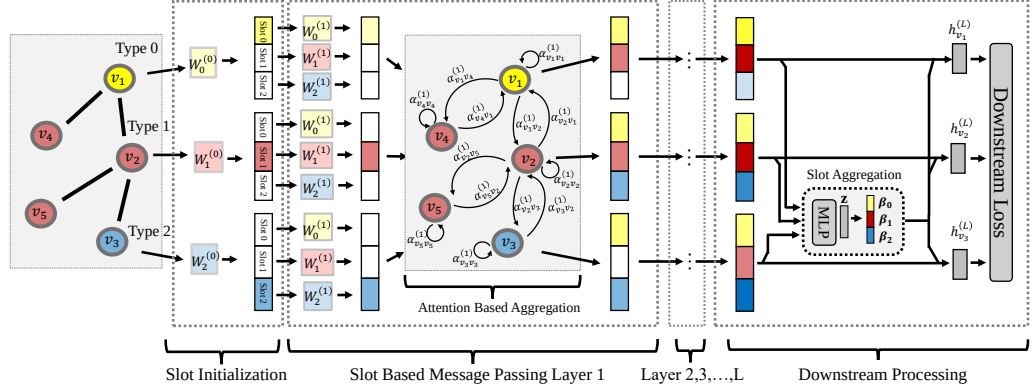


Figure 4.2: The SlotGAT architecture. (i) SlotGAT initializes every node with multiple slots corresponding to all node types (3 types in the example), and the slot for the node’s type is initialized by its transformed features, while the other slots are empty. (ii) In a slot-based message passing layer, slot representations are transformed and propagated separately, with an attention based aggregation technique. (iii) After the last L -th layer, SlotGAT includes a slot attention technique to integrate slots for downstream tasks.

handle large heterogeneous graphs with heterogeneous subgraph sampling techniques. HetGNN [156] uses random walks to sample fixed-size neighbors for nodes in different node type, and then applies RNNs for representation learning. As shown in experiments, HetGNN is inferior to our method. HetSANN [53] contains type-specific graph attention layers to aggregate local information. In experiments, SlotGAT is better than HetSANN. Recently, Lv *et al.*, [88] develop simpleHGN with several techniques on homogeneous GNNs to handle heterogeneous graphs. Space4HGNN [160] defines a unified design space for HGNNs, to exhaustively evaluate the combinations of many techniques. [131] propose a disentangled mechanism in which an aspect of a target node aggregates semantics from all neighbors for recommendation, regardless of node types. Contrarily, we learn separate semantics in different node type feature spaces, which is different.

4.4 The SlotGAT Method

The design of SlotGAT is motivated by addressing the semantic mixing issue prevalent in existing approaches, such as those relying on meta-path attention or relation-specific transformations. These methods often entangle features from different node types into a single representation space, potentially obscuring distinct semantic information. To resolve this, we propose slot-based representations to explicitly separate and preserve the unique semantics associated with different node types throughout the message passing process.

Furthermore, inspired by recent findings [88] that simple architectures incorporating essential heterogeneity information can achieve robust performance, we aim to avoid over-reliance on complex inductive biases. Instead, SlotGAT seeks a minimal yet effective solution to leverage heterogeneity: utilizing distinct slots for node types and an attention mechanism for edge types. Our empirical study confirms that this streamlined design effectively captures structural and semantic heterogeneity without unnecessary complexity.

Figure 4.2 shows the architecture of SlotGAT, consisting of node-type slot initialization, slot-based message passing layers with attention based aggregation, and a slot attention module.

Given a heterogeneous graph $\mathcal{G}$, SlotGAT creates $|\Phi|$ slots for every node (3 slots per node in Figure 4.2). The t -th slot of a node v represents the semantic representation of v with respect to node type t . For initialization, as in Figure 4.2, if slot t of v (*e.g.*, v_3) corresponds to the node type of v , $\phi(v)$ (*e.g.*, $t = 2$), then slot t is initialized by node v 's features $\mathbf{x}_v$ via a linear transformation. All other slots of v are initialized as zero (*e.g.*, slots 0, 1 of v_3 in Figure 4.2).

Then within a slot-based message passing layer of SlotGAT, given a node v , its neighbors transform and pass slot-wise messages to it. Compared with existing methods

that only pass *one* message from a neighbor u to v , SlotGAT passes $|\Phi|$ *slot messages separately* to v , and these messages do not mix with each other in intermediate layers. As illustrated in Figure 4.2, in Layer $l = 1$, for any neighbor u of v , we apply transformation by $\mathbf{W}_t^{(l)}$ independently on each slot t of u , and then pass the transformed slot messages to v for slot-wise aggregation with an attention mechanism that leverages slot representations and edge types to compute attention scores, elaborated in Section 4.4.2. Remark that the slot t of v preserves the type- t specific semantics delivered from the graph to v , regardless whether v is in type t or not. With this novel slot-based message passing in a layer, SlotGAT are able to maintain distinguishable representations of a node in a finer granularity.

After L layers of slot-based message passing in SlotGAT, every node v has $|\Phi|$ slot representations as shown in Figure 4.2. For downstream tasks, we develop a slot attention technique in Section 4.4.3 to learn slot importance scores to get final representations, which are then fed into downstream tasks. Algorithm 2 shows the pseudo code of SlotGAT.

4.4.1 Node Type Slot Initialization

We create $|\Phi|$ type-specific slots for every node. Then given a node v with type $\phi(v)$, its slot t $\mathbf{h}_v^{(0),t}$ is initialized by Eq. 4.1. Specifically, if slot t is not type $\phi(v)$, then $\mathbf{h}_v^{(0),t}$ is set to zero initially; otherwise, $\mathbf{h}_v^{(0),t}$ is initialized by the feature vector of v with a t type-specific linear transformation $\mathbf{W}_t^{(0)}$. Since the feature vectors of different types of nodes can be in different dimensions, $\mathbf{W}_t^{(0)}$ is a pre-processor to map heterogeneous node features to the same dimension d_1 .

$$\mathbf{h}_v^{(0),t} = \begin{cases} \mathbf{W}_t^{(0)} \mathbf{x}_v \in \mathbb{R}^{d_1} & \text{if } t = \phi(v), \\ \mathbf{0} \in \mathbb{R}^{d_1} & \text{if } t \neq \phi(v), \end{cases} \quad (4.1)$$

where $\mathbf{W}_t^{(0)} \in \mathbb{R}^{d_1 \times d_0^{\phi(v)}}$ is a t -type transformation matrix.

4.4.2 Slot-based Message Passing Layer

In this section, we present the slot-based message passing layer of SlotGAT with slot-specific transformations and a new attention mechanism.

In the l -th layer of SlotGAT, all t -type slots of all nodes in $\mathcal{G}$ maintain type-specific (or slot-specific) transformations $\mathbf{W}_t^{(l)}$. Given a node v with slot t representation $\mathbf{h}_v^{(l-1),t}$ from the $(l-1)$ -th layer, in current l -th layer, the representation is transformed to $\hat{\mathbf{h}}_v^{(l),t}$ by $\mathbf{W}_t^{(l)}$ in Eq. 4.2. Note that the transformation is within the feature space of node type t , and does not affect the feature spaces of other types.

$$\hat{\mathbf{h}}_v^{(l),t} = \mathbf{W}_t^{(l)} \mathbf{h}_v^{(l-1),t}, \quad (4.2)$$

where $\mathbf{W}_t^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}}$ is a slot-specific transformation.

For instance, in Figure 4.2, in the first layer, the transformation matrix $\mathbf{W}_1^{(1)}$ for slot 1 (in red) is applied to the slots 1 of all nodes. Even if the node is not in type $t = 1$, its slot 1 will be transformed by $\mathbf{W}_1^{(1)}$. Observe that, if a slot is empty, its message will still be zero vector after transformation.

In the l -th layer of SlotGAT ($0 < l \leq L$), for every slot t of every node $v \in \mathcal{V}$ ($t \in \Phi$), we perform the above type-specified linear transformation to get slot messages $\hat{\mathbf{h}}_v^{(l),t}$, which will be aggregated in the l -th layer via a generic slot-based message aggregation function *aggr*. In particular, given a node v , its slot t receives messages $\hat{\mathbf{h}}_u^{(l),t}$ from its neighbors $u \in N(v)$, which are then aggregated to update its slot t to be $\mathbf{h}_v^{(l),t}$,

$$\mathbf{h}_v^{(l),t} = \text{aggr} \left(\{ \hat{\mathbf{h}}_u^{(l),t} \}_{u \in N(v)} \right), \forall t \in \Phi, \quad (4.3)$$

where $\mathbf{h}_v^{(l),t} \in \mathbb{R}^{d_l}$, and $\hat{\mathbf{h}}_v^{(l),t} \in \mathbb{R}^{d_l}$.

Observe that in the propagation and aggregation process above, SlotGAT uses the whole graph topology to propagate slot messages. Moreover, there can be multiple options for the *aggr* function, *e.g.*, mean and max [49]. Instead of using these vanilla aggregation functions, we develop an attention mechanism to learn the aggregation

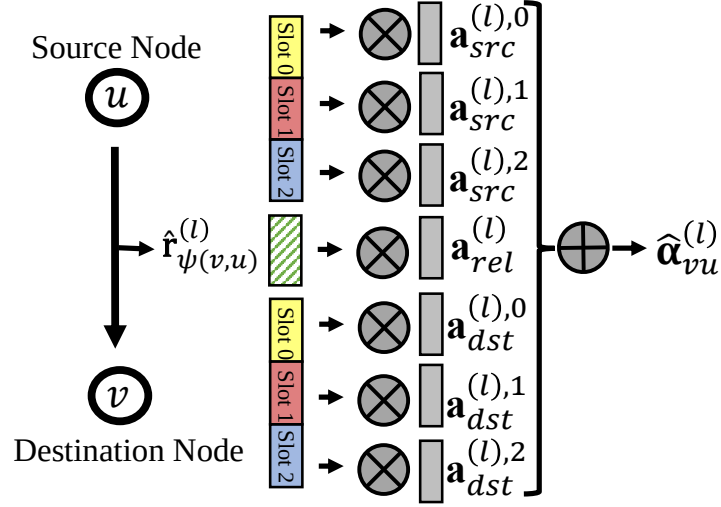


Figure 4.3: Attention mechanism in a message passing layer.

weights $\alpha_{vu}^{(l)}$ in Eq. 4.4, where the aggregation result is then passed via non-linear ReLU activation σ .

$$\mathbf{h}_v^{(l),t} = \sigma \left(\sum_{u \in N(v)} \alpha_{vu}^{(l)} \hat{\mathbf{h}}_u^{(l),t} \right), \forall t \in \Phi. \quad (4.4)$$

Now we elaborate how to get aggregation weight $\alpha_{vu}^{(l)}$. Intuitively, node u passes all its slot messages to v for all slots $t \in \Phi$. Thus, weight $\alpha_{vu}^{(l)}$ should be computed by considering all the slot messages from u to v , as well as edge types, instead of only using a specific slot t . In other words, all slot messages from node u to v share the same aggregation weight. Specifically, we develop a self-attention technique with slot message encoders and edge-type encoders to compute $\alpha_{vu}^{(l)} \in \mathbb{R}$, as illustrated in Figure 4.3. In the l -th layer, all nodes serving as the source of messages (*e.g.*, u in Figure 4.3) share a slot-specific attention vector $\mathbf{a}_{src}^{(l),t} \in \mathbb{R}^{d_l}$ for every $t \in \Phi$. Similarly, all nodes that serve as the destination to receive messages (*e.g.*, v in Figure 4.3) share a slot-specific attention vector $\mathbf{a}_{dst}^{(l),t} \in \mathbb{R}^{d_l}$ for every $t \in \Phi$. Further, let $\mathbf{a}_{rel}^{(l)} \in \mathbb{R}^{d_e}$ be an attention vector for edge types. Then the overall attention vector $\mathbf{a}^{(l)}$ is obtained by

$$\mathbf{a}^{(l)} = \left(\parallel \mathbf{a}_{src}^{(l),t} \right) \parallel \left(\parallel \mathbf{a}_{dst}^{(l),t} \right) \parallel \mathbf{a}_{rel}^{(l)}. \quad (4.5)$$

To get the edge type embedding of l -th layer, $\hat{\mathbf{r}}_{\psi(v,u)}^{(l)}$, we use a similar idea in [88]. Let

$\mathbf{r}_{\psi(v,u)} \in \mathbb{R}^{d_e}$ be a learnable representation of edge type $\psi(v, u)$, and it is randomly initialized and shared across all layers for all edges with type $\psi(v, u)$. It is transformed to get the l -th layer representation $\hat{\mathbf{r}}_{\psi(v,u)}^{(l)} = \mathbf{W}_{rel}^{(l)} \mathbf{r}_{\psi(v,u)}$, where $\mathbf{W}_{rel}^{(l)} \in \mathbb{R}^{d_e \times d_e}$ is a learnable matrix. Then we compute the attention score $\alpha_{vu}^{(l)}$ by Eq. 4.6 and 4.7. In Eq. 4.6, we first get an intermediate $\hat{\alpha}_{vu}^{(l)}$ by inner product between the corresponding attention vectors and representations, including source node's slot representations $\hat{\mathbf{h}}_u^{(l),t}$, destination node's slot representations $\hat{\mathbf{h}}_v^{(l),t}$, and edge type representations $\hat{\mathbf{r}}_{\psi(v,u)}^{(l)}$. Then we use softmax normalization and LeakyReLU to get $\alpha_{vu}^{(l)}$ in Eq. 4.7.

$$\hat{\alpha}_{vu}^{(l)} = \sum_{t \in \Phi} \langle \mathbf{a}_{dst}^{(l),t}, \hat{\mathbf{h}}_v^{(l),t} \rangle + \sum_{t \in \Phi} \langle \mathbf{a}_{src}^{(l),t}, \hat{\mathbf{h}}_u^{(l),t} \rangle + \langle \mathbf{a}_{rel}^{(l)}, \hat{\mathbf{r}}_{\psi(v,u)}^{(l)} \rangle \quad (4.6)$$

$$\alpha_{vu}^{(l)} = \frac{\exp \left(\text{LeakyReLU} \left(\hat{\alpha}_{vu}^{(l)} \right) \right)}{\sum_{u \in N(v)} \exp \left(\text{LeakyReLU} \left(\hat{\alpha}_{vu}^{(l)} \right) \right)} \quad (4.7)$$

After L slot-based message passing layers, SlotGAT outputs the representation $\mathbf{h}_v^{(L),t}$ of every slot t in every node $v \in \mathcal{V}$.

To provide stability to the training process, SlotGAT employs multi-head attention mechanism with K heads [130, 119, 57],

$$\begin{aligned} \hat{\mathbf{h}}_{v,(k)}^{(l),t} &= \mathbf{W}_{t,(k)}^{(l)} \mathbf{h}_v^{(l-1),t} \in \mathbb{R}^{d_l}, \\ \mathbf{h}_v^{(l),t} &= \parallel_{k=1,2,\dots,K} \sigma \left(\sum_{u \in N(v)} \alpha_{vu,(k)}^{(l)} \hat{\mathbf{h}}_{u,(k)}^{(l),t} \right), \end{aligned} \quad (4.8)$$

where subscript (k) represents the corresponding variables for the k -th head.

4.4.3 Slot Attention

After L layers, SlotGAT returns slot representations $\mathbf{h}_v^{(L),t}$ for $|\Phi|$ slots of nodes $v \in \mathcal{V}$. To handle the downstream tasks with objectives to be elaborated in Section 4.4.4, we explain how to leverage all slot representations of node v to get its final representation $\mathbf{h}_v$ by a slot attention technique.

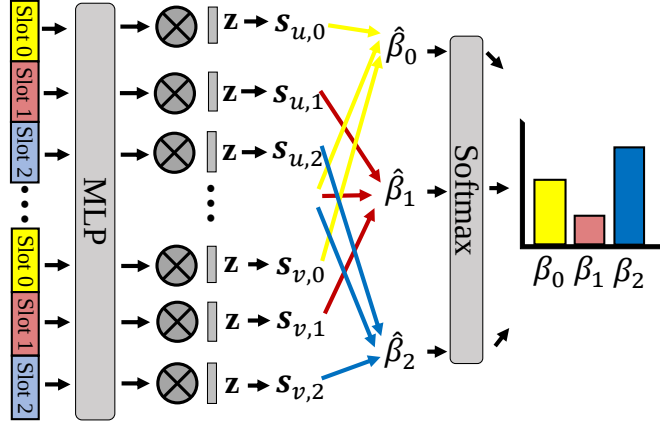


Figure 4.4: Slot attention

Given the slot representations of node v , $\mathbf{h}_v^{(L),t}, \forall t \in \Phi$, a simple way is to average them by Eq. 4.9, which however, does not differentiate slot importance.

$$\mathbf{h}_v = \frac{1}{|\Phi|} \sum_{t \in \Phi} \mathbf{h}_v^{(L),t}. \quad (4.9)$$

Another way is to stack a fully connected layer,

$$\mathbf{h}_v = \mathbf{W}_{fc} \mathbf{h}_v^{(L),t}, \quad (4.10)$$

where $\mathbf{W}_{fc} \in \mathbb{R}^{d_{task} \times (|\Phi|d_L)}$ is a learnable matrix.

But the way above is not explainable to demonstrate the importance of different slots in different tasks. Therefore, we develop a slot attention technique (illustrated in Figure 4.4), which achieves better performance than the two ways above as validated in experiments.

Specifically, in slot attention, we first apply a one-layer MLP to transform every slot representation $\mathbf{h}_v^{(L),t}$ of every node v of the L -th layer to vector $\mathbf{s}_{v,t}$,

$$\mathbf{s}_{v,t} = \tanh(\mathbf{W}_b \mathbf{h}_v^{(L),t} + \mathbf{b}_b), \quad (4.11)$$

where $\mathbf{W}_b \in \mathbb{R}^{d_s \times d_L}$, $\mathbf{b}_b \in \mathbb{R}^{d_s}$ are weights and bias.

$\mathbf{s}_{v,t}$ represents the slot semantics decoded from slot t of node v in last L -th layer. Then, for each slot t , we get its semantic importance, by computing the expected similarity

between the decoded semantics $\mathbf{s}_{v,t}$ and a learnable attention vector $\mathbf{z} \in \mathbb{R}^{d_L}$, which is shared across all slots of all nodes. For slot t , its intermediate attention score (without normalization) $\hat{\beta}_t$ is computed as follows, which considers the t -th slots of all nodes in $\mathcal{V}$. Then we apply softmax to all $\hat{\beta}_t, \forall t \in \Phi$, to get the slot attention weight β_t ,

$$\hat{\beta}_t = \frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} \langle \mathbf{z}, \mathbf{s}_{v,t} \rangle; \beta_t = \frac{\exp(\hat{\beta}_t)}{\sum_{\theta \in \Phi} \exp(\hat{\beta}_\theta)}. \quad (4.12)$$

Finally, we aggregate the representations of all slots in a node v to get its final representation $\mathbf{h}_v$. Let matrix $\mathbf{H}$ represented all representations $\mathbf{h}_v$ of all nodes $v \in \mathcal{V}$.

$$\mathbf{h}_v = \sum_{t \in \Phi} \beta_t \mathbf{h}_v^{(L),t}. \quad (4.13)$$

4.4.4 Downstream Objectives

Node Classification. The representation $\mathbf{h}_v$ is used as the final output prediction vector for every node $v \in \mathcal{V}$. Supposing that there are C classes, $\mathbf{h}_v$ is in $\mathbb{R}^{d_C}$. Let $\mathbf{H}_{v,c}$ be the probability that node v is predicted to have class label c . Denote $\mathbf{Y}$ as the groundtruth label matrix whose entries $\mathbf{Y}_{v,c} = 1$ if node v has label c ; otherwise $\mathbf{Y}_{v,c} = 0$. A cross-entropy loss function is used to train the model over labeled training nodes $\mathcal{L}_{train} \subset \mathcal{V}$: $\mathbf{L}_{NC}(\mathbf{H}, \mathcal{G}) = -\frac{1}{|\mathcal{L}_{train}|} \sum_{v \in \mathcal{L}_{train}} \sum_{c \in [C]} \mathbf{Y}_{v,c} \log(\mathbf{H}_{v,c})$.

Link Prediction. Link prediction is binary classification on node pairs. Positive pairs $\mathcal{E}_{pos} \subset \mathcal{V} \times \mathcal{V}$ are the edges actually in $\mathcal{E}$, and negative pairs $\mathcal{E}_{neg} \subset \mathcal{V} \times \mathcal{V}$ are non-existing ones. $\mathcal{E}_{pos}$ and $\mathcal{E}_{neg}$ together are training samples. The model is expected to distinguish node pairs in test set whether it should be positive or negative. Given two nodes v, u with representations $\mathbf{H}_v$ and $\mathbf{H}_u$ respectively, a common method is to decode them to get a similarity score, which is then fed into a binary cross-entropy loss: $\mathbf{L}_{LP}(\mathbf{H}, \mathcal{G}) = -\sum_{(v,u) \in \mathcal{E}_{pos}} \log[\text{Decoder}(\mathbf{H}_v, \mathbf{H}_u)] - \sum_{(v,u) \in \mathcal{E}_{neg}} \log[1 - \text{Decoder}(\mathbf{H}_v, \mathbf{H}_u)]$. Two decoders are considered: dot product and DistMult [142].

Algorithm 2 SlotGAT

Input: Heterogeneous graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \phi, \psi)$, number of heads K , number of layers L , number of epochs T

Output: The learned representations $\mathbf{H}$.

Generate initial parameters for all learnable parameters.

for each epoch $i = 0, 1, 2, \dots, T$ **do**

 Perform node type based slot initialization to get $\mathbf{h}_v^{(0),t}$;

for each layer $l = 0, 1, 2, \dots, L$ **do**

for each head $k = 0, 1, 2, \dots, K$ **do**

 Construct slot messages $\hat{\mathbf{h}}_{v,(k)}^{(l),t} = \mathbf{W}_{t,(k)}^{(l)} \mathbf{h}_v^{(l-1),t}$;

 Compute attention $\alpha_{vu,(k)}^{(l)}$;

end for

 Aggregate the messages with attention

$\mathbf{h}_v^{(l),t} \leftarrow \prod_{k=1,2,\dots,K} \sigma(\sum_{u \in N(v)} \alpha_{vu,(k)}^{(l)} \hat{\mathbf{h}}_{u,(k)}^{(l),t})$;

end for

 Compute semantics vectors $\mathbf{s}_{v,t} \leftarrow \text{MLP}(\mathbf{h}_v^{(L),t})$;

 Compute slot attention β_t ;

 Aggregate the slots $\mathbf{h}_v \leftarrow \sum_{t \in \Phi} \beta_t \mathbf{h}_v^{(L),t}$;

end for

Link Prediction Decoders Given $\mathbf{H}_v$ and $\mathbf{H}_u$, dot product decoder first computes dot product and then applies a sigmoid activation function: $\text{Decoder}(\mathbf{H}_v, \mathbf{H}_u) = \sigma(\langle \mathbf{H}_v, \mathbf{H}_u \rangle)$.

DistMult requires a learnable square weight matrix $\mathbf{W}_{\psi(v,u)} \in \mathbb{R}^{d_L \times d_L}$ for edge type $\psi(v, u)$. Then a bi-linear form between $\mathbf{H}_v, \mathbf{H}_u$ is calculated as: $\text{Decoder}(\mathbf{H}_v, \mathbf{H}_u) = \sigma(\mathbf{H}_v^\top \mathbf{W}_{\psi(v,u)} \mathbf{H}_u)$. The choice of dot product or DistMult is regarded as a part of hyper-tuning process [88].

4.4.5 Analysis

We conduct theoretical analysis to motivate the design of slots for different node types to learn different semantics, which mitigates the semantic mixing issue. We adopt the row-normalized Laplacian model in [66] for analysis, and the empirical property of attention models are similar to spectra-based graph convolution methods with row-normalization [16].

Let A be the adjacency matrix and D be the degree matrix of heterogeneous graph $\mathcal{G}$, and define $L_{rw} = D^{-1}(D - A)$. The spectral graph convolution operator G is $G = I - \alpha L_{rw}$, where $\alpha \in (0, 1]$. Suppose that $\mathcal{G}$ has k maximal connected components (CC) $\{CC_i\}_{i=1}^k$. Let $cc(j)$ denote the CC where node v_j is, and $|cc(j)|$ be the size of the CC. Considering the slot initialization in Section 4.4.1, let $\mathbf{X}^{(t)} \in \mathbb{R}^{n \times d_0^t}$ be the initial feature matrix of slots t of all n nodes in $\mathcal{G}$, and the i -th row vector $\mathbf{x}_i^{(t)}$ be the initial feature vector of slot t of node v_i . Only nodes in type t have nonzero $\mathbf{x}_i^{(t)}$ in $\mathbf{X}^{(t)}$; otherwise, $\mathbf{x}_i^{(t)} = 0$. With self-loop added to every node, there is no bipartite component in the graph. Then inspired by [78], we derive Theorem 1, with proof in Section 4.4.6.

Theorem 1. *Given a heterogeneous graph $\mathcal{G}$, for the t -slot feature matrix $\mathbf{X}^{(t)} \in \mathbb{R}^{n \times d_0^t}$ in which only nodes of type t have non-zero features, if the graph has no bipartite components, after infinite number of convolution operations G , the slot t representation of node v_i is*

$$\left[\lim_{\ell \rightarrow +\infty} G^\ell \mathbf{X}^{(t)} \right]_i = \frac{1}{|cc(i)|} \sum_{\forall v_j \in cc(i), \phi(v_j)=t} \mathbf{x}_j^{(t)}.$$

4.4.6 Proof of Theorem 1

Proof. The indication vector for the i -th CC is denoted by $\mathbf{1}^{(i)} \in \mathbb{R}^n$. This vector indicates whether a node is in the maximal connected component CC_i , i.e.,

$$\mathbf{1}_j^{(i)} = \begin{cases} 1, v_j \in CC_i \\ 0, v_j \notin CC_i \end{cases} \quad (4.14)$$

If a graph has no bipartite components, the eigenvalues are all in $[0, 2)$ [25]. The eigenspace of L_{rw} corresponding to eigenvalue 0 is spanned by $\{\mathbf{1}^{(i)}\}_{i=1}^k$ [122]. For $\alpha \in (0, 1]$, the eigenvalues of $(I - \alpha L_{rw})$ fall into $(-1, 1]$, and the eigenspace of eigenvalue 1 is spanned by $\{\mathbf{1}^{(i)}\}_{i=1}^k$.

Denote $(I - \alpha L_{rw}) = G = P \text{diag}\{\lambda_1, \lambda_2, \dots, \lambda_n\} P^\top$ as the diagonalization of symmetric matrix G , where λ_i is the eigenvalues and the i -th column of matrix P is the corresponding eigenvector. We compute

$$\begin{aligned} & [\lim_{\ell \rightarrow +\infty} (I - \alpha L_{rw})^\ell \mathbf{X}^{(t)}]_i \\ &= [\lim_{\ell \rightarrow +\infty} G^\ell \mathbf{X}^{(t)}]_i \\ &= [P \text{diag}\{\lim_{\ell \rightarrow +\infty} \lambda_1^\ell, \lim_{\ell \rightarrow +\infty} \lambda_2^\ell, \dots, \lim_{\ell \rightarrow +\infty} \lambda_n^\ell\} P^\top \mathbf{X}^{(t)}]_i \end{aligned} \quad (4.15)$$

Since all eigenvalues $\lambda_i \in (-1, 1]$, only the eigenvalues equal to one could avoid shrinking to 0. Thus,

$$\begin{aligned} & [P \text{diag}\{\lim_{\ell \rightarrow +\infty} \lambda_1^\ell, \lim_{\ell \rightarrow +\infty} \lambda_2^\ell, \dots, \lim_{\ell \rightarrow +\infty} \lambda_n^\ell\} P^\top \mathbf{X}^{(t)}]_i \\ &= \sum_{j=1}^n \sum_{o=1}^n (\lim_{\ell \rightarrow +\infty} \lambda_o^\ell) P_{io} P_{jo} \mathbf{x}_j^{(t)} \\ &= \sum_{j=1}^n \sum_{\substack{1 \leq o \leq n, \\ \lambda_o = 1}} P_{io} P_{jo} \mathbf{x}_j^{(t)} \end{aligned} \quad (4.16)$$

Since the eigenspace of matrix G corresponding to eigenvalue 1 are spanned by $\{\mathbf{1}^{(i)}\}_{i=1}^k$ [122], and all of the eigenvectors within the matrix P 's columns are orthogonal to each other, we can see that each column $P_{:,o}$ corresponding to eigenvalue

1 (the count of this kind of vectors is equal to k) will be one of the vectors in the set $\{\frac{1}{\sqrt{|CC_i|}} \cdot \mathbf{1}^{(i)}\}_{i=1}^k$. These k vectors all have 2-norm equal to 1, where $|CC_i|$ is the size of the i -th CC. Thus if $\lambda_o = 1$, $P_{:,o}$ is one of the vectors in set $\{\frac{1}{\sqrt{|CC_i|}} \mathbf{1}^{(i)}\}_{i=1}^k$. Then $P_{io}P_{jo} = \frac{1}{|cc(i)|} \iff v_j \in cc(i)$ (node v_i and node v_j in the same CC), where $|cc(i)|$ is the size of the CC which node i is in, otherwise $P_{io}P_{jo} = 0$. Therefore,

$$\begin{aligned} [\lim_{\ell \rightarrow +\infty} G^\ell \mathbf{X}^{(t)}]_i &= \sum_{j=1}^n \sum_{\substack{1 \leq o \leq n, \\ \lambda_o = 1}} P_{io}P_{jo} \mathbf{x}_j^{(t)} \\ &= \sum_{v_j \in cc(i)} \frac{1}{|cc(i)|} \mathbf{x}_j^{(t)}. \end{aligned} \quad (4.17)$$

Since only nodes with type t have non-zero features, Eq. 4.17 is re-written as:

$$\begin{aligned} [\lim_{\ell \rightarrow +\infty} G^\ell \mathbf{X}^{(t)}]_i &= \sum_{v_j \in cc(i)} \frac{1}{|cc(i)|} \mathbf{x}_j^{(t)} \\ &= \sum_{\substack{v_j \in cc(i), \\ \phi(v_j) = t}} \frac{1}{|cc(i)|} \mathbf{x}_j^{(t)}. \end{aligned} \quad (4.18)$$

□

As proved in the theorem, after infinite steps, the representation of the slot t in node v_i converges to the average feature vector of all the nodes in the same type t in $cc(i)$. Since the semantics of the nodes in a different node type t' with initial features $\mathbf{X}^{(t')}$, are usually very different from $\mathbf{X}^{(t)}$, the representations for slots t' will apparently converge to a different state. Theorem 1 proves that the slots of the nodes within the *same* connected component $cc(i)$ converge to *different* states $\frac{1}{|cc(i)|} \sum_{v_j \in cc(i), \phi(v_j)=t} \mathbf{x}_j^{(t)}$ that are relevant to node type t feature spaces, which explains the power of SlotGAT to mitigate the semantic mixing issue. On the other hand, [78] prove that all node representations in a homogeneous connected component converge to the same state. SlotGAT performs L layers ($\ll +\infty$), and thus the slot representations output by SlotGAT, $\mathbf{h}_{v_i}^{(L),t}$ are semantically different from $\mathbf{h}_{v_i}^{(L),t'}$ for slots t and t' of node v_i in $\mathcal{G}$. We further provide visualization results of slot representations in Section 4.5.4 to validate that the slot-based message passing in SlotGAT captures rich semantics.

Table 4.1: Statistics of Benchmark Datasets.

<i>Node Classification</i>	#Nodes	#Node Types	#Edges	#Edge Types	Target	#Classes
DBLP	26,128	4	239,566	6	author	4
IMDB	21,420	4	86,642	6	movie	5
ACM	10,942	4	547,872	8	paper	3
Freebase	180,098	8	1,057,688	36	book	7
PubMed_NC	63,109	4	244,986	10	disease	8
<i>Link Prediction</i>					Target	
LastFM	20,612	3	141,521	3	user-artist	
PubMed_LP	63,109	4	244,986	10	disease-disease	

Complexity. We then provide the complexity analysis of SlotGAT, by following a similar fashion as [119]. Let m , n and $|\Phi|$ be the number of nodes, edges, and node types, respectively, and assume that all representations in all layers have the same dimension d . In one SlotGAT layer (Section 4.4.2), (i) feature transformation in Eq. 4.2 needs $O(nd^2|\Phi|)$; (ii) attention computation in Eq. 4.6 and Eq. 4.7 needs $O(md|\Phi|)$. Hence, the complexity of a single layer is $O(nd^2|\Phi| + md|\Phi|)$, which is on par with baseline methods if factor $|\Phi|$ is regarded as a constant. According to [119], applying K heads multiplies the storage and parameters, while the computation of individual heads is fully independent and parallelized. The computation of Eq. 4.1 in Section 4.4.1 is only performed once for all n nodes before the L -layer message passing, and it needs time $O(ndd_0|\Phi|)$, which is $O(nd^2|\Phi|)$ if the input feature dimension d_0 is d as well. The slot attention in Section 4.4.3 is also computed only once after message passing, and its complexity is $O(ndd_s|\Phi|)$, which is $O(nd^2|\Phi|)$ if the embedding dimension of slot attention d_s is also d . The number of layers L is usually small as a constant, e.g., 3. Hence, the complexity of SlotGAT is $O(nd^2|\Phi| + md|\Phi|)$.

Table 4.2: Node classification results with mean and standard deviation of Macro-F1/Micro-F1 (Part I). Vacant positions ("–") mean out of memory. Best is in bold, and runner up is underlined.

	DBLP		IMDB		ACM	
	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1
RGCN	91.52±0.50	92.07±0.50	58.85±0.26	62.05±0.15	91.55±0.74	91.41±0.75
HAN	91.67±0.49	92.05±0.62	57.74±0.96	64.63±0.58	90.89±0.43	90.79±0.43
DisenHAN	93.66±0.39	94.18±0.36	63.40±0.49	<u>67.48±0.45</u>	92.52±0.33	92.45±0.33
GTN	93.52±0.55	93.97±0.54	60.47±0.98	65.14±0.45	91.31±0.70	91.20±0.71
RSHN	93.34±0.58	93.81±0.55	59.85±3.21	64.22±1.03	90.50±1.51	90.32±1.54
HetGNN	91.76±0.43	92.33±0.41	48.25±0.67	51.16±0.65	85.91±0.25	86.05±0.25
MAGNN	93.28±0.51	93.76±0.45	56.49±3.20	64.67±1.67	90.88±0.64	90.77±0.65
HetSANN	78.55±2.42	80.56±1.50	49.47±1.21	57.68±0.44	90.02±0.35	89.91±0.37
HGT	93.01±0.23	93.49±0.25	63.00±1.19	67.20±0.57	91.12±0.76	91.00±0.76
GCN	90.84±0.32	91.47±0.34	57.88±1.18	64.82±0.64	92.17±0.24	92.12±0.23
GAT	93.83±0.27	93.39±0.30	58.94±1.35	64.86±0.43	92.26±0.94	92.19±0.93
simpleHGN	94.01±0.27	94.46±0.20	<u>63.53±1.36</u>	67.36±0.57	<u>93.42±0.44</u>	<u>93.35±0.45</u>
space4HGNN	<u>94.24±0.42</u>	<u>94.63±0.40</u>	61.57±1.19	63.96±0.43	92.50±0.14	92.38±0.10
SlotGAT	94.95±0.20	95.31±0.19	64.05±0.60	68.54±0.33	93.99±0.23	94.06±0.22

4.5 Experiments

All experiments are conducted on a machine powered by an Intel(R) Xeon(R) E5-2603 v4 @ 1.70GHz CPU, 131GB RAM, and a Nvidia Geforce 3090 Cards with Cuda version 11.3. Source codes of all competitors are obtained from respective authors.

4.5.1 Experiment Settings

Datasets. Table 4.1 reports the statistics of benchmark datasets widely used in [88, 130, 160, 156, 153, 144]. The datasets cover various domains, including academic graphs (*e.g.*, DBLP, ACM), information graphs (*e.g.*, IMDB, LastFM, Freebase), and medical biological graph (*e.g.*, PubMed). For node classification, each dataset contains a target node type, and all nodes with the target type are the nodes for classification [88]. PubMed_LP is the same as PubMed_NC, but for link prediction.

Table 4.3: Node classification results with mean and standard deviation of Macro-F1/Micro-F1 (Part II). Vacant positions ("") mean out of memory. Best is in bold, and runner up is underlined.

	PubMed_NC		Freebase	
	Macro-F1	Micro-F1	Macro-F1	Micro-F1
RGCN	18.02±1.98	20.46±2.39	46.78±0.77	58.33±1.57
HAN	15.43±2.41	24.88±2.27	21.31±1.68	54.77±1.40
DisenHAN	41.71±4.43	50.93±4.25	-	-
GTN	-	-	-	-
RSHN	-	-	-	-
HetGNN	21.86±3.21	29.93±3.51	-	-
MAGNN	-	-	-	-
HetSANN	-	-	-	-
HGT	<u>47.50±6.34</u>	<u>51.86±4.85</u>	29.28±2.52	60.51±1.16
GCN	9.84±1.69	21.16±2.00	27.84±3.13	60.23±0.92
GAT	24.89±8.47	34.65±5.71	40.74±2.58	65.26±0.80
simpleHGN	42.93±4.01	49.26±3.32	<u>47.72±1.48</u>	<u>66.29±0.45</u>
space4HGNN	45.53±4.64	49.76±3.92	41.37±4.49	65.66±4.94
SlotGAT	47.79±3.56	53.25±3.40	49.68±1.97	66.83±0.30

Each link prediction dataset has a target edge type, and edges in the target edge type are the target for prediction.

Dataset Descriptions

For all of the benchmark datasets, one could access them in online platform HGB¹. In the following, we provide the descriptions of these datasets. The detailed data statistics, including the number of nodes in every type and the number of edges in every type.

- **DBLP**² is a bibliography website of computer science. There are 4 node types, including authors, papers, terms, and venues, as well as 6 edge types. The edge types include paper-term, paper-term, paper-venue, paper-author, term-paper, and venue-paper. The target is to predict the class labels of authors. The

¹<https://www.biendata.xyz/hgb/>

²<http://web.cs.ucla.edu/~yzsun/data/>

Table 4.4: Data Split of Node Classification Datasets.

	#Nodes	#Training	#Validation	#Testing
DBLP	26,128	974	243	2,840
IMDB	21,420	1,097	274	3,159
ACM	10,942	726	181	2,118
Freebase	180,098	1,909	477	4,446
PubMed_NC	63,109	295	73	86

Table 4.5: The searched hyper-parameters for SlotGAT with best performances on various datasets (Part 1).

Datasets	AttDim	lr	wd	dropAttn	droptFeat	featType	Epoch
DBLP	32	1e-4	1e-3	0.5	0.5	1	300
IMDB	3	1e-4	1e-3	0.2	0.8	1	300
ACM	32	1e-3	1e-4	0.8	0.8	1	300
pubmed_NC	3	5e-3	1e-3	0.8	0.2	1	300
Freebase	8	5e-4	1e-3	0.5	0.5	2	300
LastFM	64	5e-4	1e-4	0.9	0.2	2	1000
pubmed_LP	32	1e-3	1e-4	0.5	0.5	2	1000

Table 4.6: The searched hyper-parameters for SlotGAT with best performances on various datasets (Part 2).

Datasets	hiddenDim	Layers	Heads	Decoder	EdgeDim
DBLP	64	4	8	-	64
IMDB	128	3	8	-	64
ACM	64	2	8	-	64
pubmed_NC	128	2	8	-	64
Freebase	16	2	8	-	0
LastFM	64	8	2	DotProduct	64
pubmed_LP	64	4	2	DistMult	64

classes are database, data mining, AI, and information retrieval.

- **IMDB**³ is a website about movies. There are 4 node types: movies, directors, actors, and keywords. A movie can have multiple class labels. The 6 edge types include movie-director, director-movie, movie-actor, actor-movie, movie-keyword, and keyword-movie. There are 5 classes: action, comedy, drama, romance, and thriller. Movies are the targets to classify.
- **ACM** is a citation network [130], containing node types of authors, papers, terms, and subjects. The edge types include paper-cite-paper, paper-ref-paper, paper-author, author-paper, paper-subject, subject-paper, paper-term, and term-paper. The node classification target is to classify papers into 3 classes: database, wireless communication, and data mining.
- **Freebase** [7] is a large knowledge graph with 8 node types, including book, film, music, sports, people, location, organization, and business, and 36 edge types. The target is to classify books in 7 classes: scholarly work, book subject, published work, short story, magazine, newspaper, journal, and poem.
- **LastFM** ⁴ is an online music website. There are 3 node types: user, artist, and tag. And it has 3 edge types: user-artist, user-user, artist-tag. The link prediction task aims to predict the edges between users and artists.
- **PubMed**⁵ is a biomedical literature library. We use the data constructed by [144]. The node types are gene, disease, chemical, and species. The 10 edge types contain gene-and-gene, gene-causing-disease, disease-and-disease, chemical-in-gene, chemical-in-disease, chemical-and-chemical, chemical-in-species, species-with-gene, species-with-disease, and species-and-species. The target of the node classification task is to predict the disease into eight categories with

³<https://www.kaggle.com/karrimba/movie-metadatacsv>

⁴<https://grouplens.org/datasets/hetrec-2011/>

⁵<https://pubmed.ncbi.nlm.nih.gov>

class labels from [144]. The target of link prediction is to predict the existence of edges between genes and diseases.

Node Classification Data Split Following [88], we have the training and validation ratio 8:2, while keeping fixed testing test, and the statistics of testing data, as well as training and validation data are listed in Table 4.4.

Baselines. We compare with HAN [130], DisenHAN [131], GTN [153], MAGNN [41], HetGNN [156], HGT [57], RGCN [104], RSHN[169], HetSANN [53], simpleHGN [88], and Space4HGNN [160]. We also compare with GCN [66] and GAT [119].

Meta-paths Used in Baselines We provide the meta-paths used in baselines in Table 4.7, which are widely adopted in [88, 41, 130].

Evaluation Settings. For node classification, following [88], we split labeled training set into training and validation with ratio 80% : 20%, while the testing data are fixed with detailed numbers in Table 4.4. For link prediction, we adopt ratio 81% : 9% : 10% to divide the edges into training, validation, and testing. For each dataset, we repeat experiments on 5 random splits and then report the average and standard deviation. For node classification, we report the averaged Macro-F1 and Micro-F1 scores. We also conduct paired t-tests [111, 68] to evaluate the statistical significance of SlotGAT. For link prediction, we report MRR (mean reciprocal rank) and ROC-AUC. Link prediction needs negative samples (*i.e.*, non-existence edges) for test, which are generated within 2-hop neighbors of a node [88].

Metrics Here we illustrate how we compute the four metrics: Macro-F1, Micro-F1, ROC-AUC and MRR.

Macro-F1: The macro F1 score is computed using the average of all the per-class F1 scores. Denote Precision_c as the precision of class c , and Recall_c as the recall of class c . We have

$$\text{Macro-F1} = \frac{1}{C} \sum_{c \in [C]} \frac{2\text{Precision}_c \cdot \text{Recall}_c}{\text{Precision}_c + \text{Recall}_c}. \quad (4.19)$$

Table 4.7: Meta-paths used in our experiments.

Dataset	Meta-paths	Meaning
DBLP	APA, APTPA, APVPA	A: Author, P: Paper, T: Term, V: Venue
IMDB	MDM, MAM, DMD, DMAMD, AMA, AMDMA, MKM	M: Movie, D: Director, A: Actor, K: Keyword
ACM	PAP, PSP, PcPAP, PcPSP, PrPAP, PrPSP, PTP	P: Paper, A: Author, S: Subject, T: Term, c: citation relation, r: reference relation
Freebase	BB, BFB, BLMB, BPB, BPSB, BOFB, BUB	B: Book, F: Film, L: Location, M: Music, P: Person, S: Sport, O: Organization, U: bUsiness
LastFM	UU, UAU, UATAU, AUA, ATA, AUUA	U: User, A: Artist, T: Tag
PubMed	DD, DGGD, DCCD, DSSD	D: Disease, G: Gene, C: Chemical, S: Species

Micro-F1: The Micro-F1 score directly uses the total precision and recall scores. With the Precision and Recall scores of all nodes regardless of their classes, we have

$$\text{Micro-F1} = \frac{2\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \quad (4.20)$$

ROC-AUC: The Area Under the Curve (AUC) score is calculated from Receiver

Operating Characteristic (ROC). Define the True Positive Rate (TPR) as $\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$, where TP, FN represent true positive and false negative respectively. The False Positive Rate (FPR) is calculated as $\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$, where FP, TN represent false positive and true negative respectively. One can see that, TPR and FPR scores vary if we change the classification threshold. One can also prove that, there exists a one-to-one function f such that under every threshold: $f(\text{FPR}) = \text{TPR}$. Then the ROC-AUC score is the area under the graph of this function f :

$$\text{ROC-AUC} = \int_{x \in [0,1]} f(x). \quad (4.21)$$

MRR: Mean reciprocal rank (MRR) is a metric usually used in recommendation system. Here we use it to evaluate performance in link prediction as in [88]. For each node v involved in concerned edges, *i.e.*, positive and negative edges to be evaluated, one can sort its all related nodes u , *i.e.*, connected by these positive or negative nodes, by the similarities of their computed embeddings. Then, denote the minimal rank of the first occurred nodes u connected with positive links $(v, u) \in \mathcal{E}_{pos}$ in the above sorted nodes list as R_v , we have the MRR as the mean of reciprocal of this minimal rank over all the involved nodes $\{(v, u) \in \mathcal{E}_{pos} \text{ or } \mathcal{E}_{neg}\}$.

Hyper-parameter Search Space. We search learning rate within $\{1, 5\} \times \{1e^{-5}, 1e^{-4}, 1e^{-3}, 1e^{-2}\}$, weight decay rate within $\{1, 5\} \times \{1e^{-5}, 1e^{-4}, 1e^{-3}\}$, dropout rate for features within $\{0.2, 0.5, 0.8, 0.9\}$, dropout rate for connections within $\{0, 0.2, 0.5, 0.8, 0.9\}$, and number of hidden layers L within $\{2, 3, 4, 5, 6\}$. We use the same dimension of hidden embeddings across all layers d_l within $\{32, 64, 128\}$. We search the number of epochs within the range of $\{40, 300, 1000\}$ with early stopping patience 40, and dimension d_s of slot attention vector within the range of $\{3, 8, 32, 64\}$. Following [88], for input feature type, we use $\text{feat} = 0$ to denote the use of all given features, $\text{feat} = 1$ to denote using only target node features (zero vector for others), and $\text{feat} = 2$ to denote all nodes with one-hot features. For node classification, we use $\text{feat} = 1$ and set the number of attention heads K to be 8. For link prediction, we use $\text{feat} = 2$ and set

Table 4.8: Link prediction results with mean and standard deviation of ROC-AUC/MRR. Vacant positions ("–") means out of memory.

	LastFM		PubMed_LP	
	ROC-AUC	MRR	ROC-AUC	MRR
RGCN	57.21±0.09	77.68±0.17	78.29±0.18	90.26±0.24
DisenHAN	57.37±0.2	76.75±0.28	73.75±1.13	85.61±2.31
HetGNN	62.09±0.01	83.56±0.14	73.63±0.01	84.00±0.04
MAGNN	56.81±0.05	72.93±0.59	–	–
HGT	54.99±0.28	74.96±1.46	80.12±0.93	90.85±0.33
GCN	59.17±0.31	79.38±0.65	80.48±0.81	90.99±0.56
GAT	58.56±0.66	77.04±2.11	78.05±1.77	90.02±0.53
simpleHGN	<u>67.59±0.23</u>	<u>90.81±0.32</u>	<u>83.39±0.39</u>	<u>92.07±0.26</u>
space4HGNN	66.89±0.69	90.77±0.32	81.53±2.51	90.86±1.02
SlotGAT	70.33±0.13	91.72±0.50	85.39±0.28	92.22±0.28

K to be 2. Link prediction has two decoders, dot product and DistMult, which are also searchable. The search space and strategies of all baselines follow [88, 160]. The searched hyper parameters are in Section 4.5.2.

4.5.2 Searched Hyper-parameters

To facilitate the re-producibility of this work, we present the searched hyper-parameters of SlotGAT in Table 4.5 and Table 4.6 on different datasets. Note that due to the high computational cost in dataset Freebase, we set the dimension $d_e = 0$ of attention vector for edge types in this dataset, while $d_e = 64$ in all other datasets. Moreover, three tricks are used in the implementation of SlotGAT as suggested in [88]: residual connection, attention residual and using hidden embedding in middle layers for link prediction tasks.

4.5.3 Evaluation Results

Node Classification. Table 4.2 and Table 4.3 report the node classification results of all methods on five datasets, with mean Macro-F1 and Micro-F1 and their standard deviation. The overall observation is that SlotGAT consistently outperforms all baselines for both Macro-F1 and Micro-F1 on all datasets, often by a significant margin, which validates the power of SlotGAT. For instance, in IMDB that is a multi-label classification dataset, the Micro-F1 of SlotGAT is 68.54%, while that of the best competitor is 67.48%; moreover, SlotGAT has a smaller standard deviation 0.33, compared to 0.57 of the competitor. On the largest Freebase, SlotGAT achieves 49.68% Macro-F1, while the best competitor has 47.72%. On PubMed_NC, SlotGAT has higher Micro-F1 53.25% compared with 51.86% of HGT. On DBLP and ACM, where all methods are with relatively high performance, SlotGAT still achieves the best performance. The superior performance of SlotGAT on node classification indicates that SlotGAT with slot-based message passing mechanism is able to learn richer representations to preserve the node heterogeneity in various heterogeneous graphs.

Link Prediction. Table 4.8 reports the link prediction results on LastFM and PubMed_LP, with mean ROC-AUC and MRR and their standard deviations. SlotGAT achieves the best performance on both datasets over both metrics. For instance, on LastFM, SlotGAT is with ROC-AUC 70.33%, 2.74% higher than simpleHGN, the best competitor, and also the MRR of SlotGAT is 91.72%, while that of simpleHGN is 90.81%. On PubMed_LP, SlotGAT is with ROC-AUC 85.39%, 2% higher than simpleHGN. The performance of SlotGAT for link prediction again reveals the effectiveness of the proposed slot-based message passing mechanism and attention techniques to avoid semantic mixing issue and preserve pairwise node relationships for link prediction.

Table 4.9: Statistical Significance Test (Node Classification)

	DBLP	IMDB	ACM	PubMed_NC	Freebase
micro-F1	0.000244	0.000111	0.00713	0.00102	0.0421
macro-F1	0.000209	0.160	0.00783	0.000192	0.156

Table 4.10: Statistical Significance Test (Link Prediction)

	LastFM	PubMed_LP
ROC-AUC	0.0252	0.00801
MRR	0.0445	0.0261

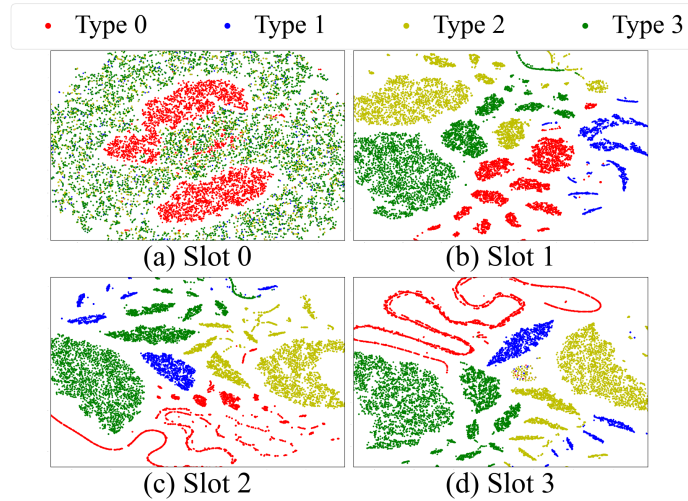


Figure 4.5: Tsne visualization on all slot representations of 2-nd SlotGAT layer on IMDB

4.5.4 Model Analysis

Slot Visualization. We adopt Tsne [117] to visualize the slot representations of SlotGAT. Figure 4.5 reports the slot visualization of the 2-nd SlotGAT layer on IMDB. Figure 4.5(a) is the visualization of the 0-th slots (in type-0 feature space) of all nodes with colors representing their own types. Figure 4.5(b), (c), (d) are the visualization of 1-st, 2-nd, and 3-rd slots of all nodes respectively. First, observe that the 4 slots of a node (for all nodes in a graph) learn radically different representations,

Table 4.11: Ablation for Slot Attention (Macro-F1/Micro-F1) on node classification. Best is in bold, and runner up is underlined.

Dataset	SlotGAT	SlotGAT (average)	SlotGAT (target slot)	SlotGAT (last fc)
DBLP (Macro-F1)	94.95±0.20	94.52±0.31	87.12±8.44	<u>94.72±0.22</u>
DBLP (Micro-F1)	95.31±0.19	94.93±0.25	87.73±7.91	<u>95.11±0.21</u>
IMDB (Macro-F1)	64.05±0.60	62.71±1.10	39.35±18.9	<u>63.15±0.53</u>
IMDB (Micro-F1)	68.54±0.33	67.56±0.42	48.91±11.2	<u>68.16±0.31</u>
ACM (Macro-F1)	93.99±0.23	93.43±0.50	93.63±1.03	<u>93.91±0.35</u>
ACM (Micro-F1)	94.06±0.22	93.35±0.43	93.50±1.15	<u>93.99±0.35</u>
PubMed_NC (Macro-F1)	<u>47.79±3.56</u>	47.18±4.16	44.60±4.14	48.24±5.37
PubMed_NC (Micro-F1)	<u>53.25±3.40</u>	52.61±3.23	50.54±4.65	53.72±2.88
Freebase (Macro-F1)	<u>49.68±1.97</u>	47.90±1.61	48.77±2.86	51.63±0.75
Freebase (Micro-F1)	<u>66.83±0.30</u>	67.01±0.90	66.38±0.50	66.68±0.42

Table 4.12: Statistical Significance Test between SlotGAT and its variants w.r.t. Micro-F1.

	DBLP	IMDB	ACM	PubMed	Freebase
SlotGAT (average)	0.0090	0.0362	0.0379	0.0226	0.3054
SlotGAT (target slot)	0.0129	0.1014	0.0300	0.0248	0.1988
SlotGAT (last fc)	0.0832	0.6714	0.2840	0.1588	0.2571

as the visualizations in the four figures exhibit different patterns. For instance, in Figure 4.5(a), 0-th slots can distinguish nodes in type 0 (red) from others, but cannot distinguish nodes of types 1, 2, 3. On the other hand, in Figure 4.5(b), 1-st slots can distinguish nodes of all types in 0, 1, 2, 3. Similar observations can be made in Figures 4.5(c) (d). The visualization in Figure 4.5 validates that SlotGAT is able to learn richer semantics in a finer granularity and also preserve the semantic differences of different node-type feature spaces, which explains the superior performance of SlotGAT. Additional visualization on DBLP is provided in Section 4.6.1.

Statistical Significance Test. In Table 4.9, we report the *p-values* of paired t-tests

Table 4.13: Statistical Significance Test between SlotGAT and its variants w.r.t. Macro-F1.

	DBLP	IMDB	ACM	PubMed	Freebase
SlotGAT (average)	0.0042	0.0325	0.0359	0.1833	0.2611
SlotGAT (target slot)	0.0304	0.1074	0.0372	0.1984	0.1160
SlotGAT (last fc)	0.0420	0.6952	0.2748	0.9704	0.2114

between SlotGAT and the best baseline simpleHGN, *w.r.t.*, micro-F1 and macro-F1 for node classification. In Table 4.10 we report the p-values on link prediction *w.r.t.* ROC-AUC and MRR. P-value below 0.05 means statistical significance. Observe that most p-values are below 0.05 except two p-values, indicating the improvement of SlotGAT over simpleHGN is statistically significant and robust.

Slot Attention Ablation. We ablate the slot attention technique in Section 4.4.3 that integrates all slots of a node, and compare with other ways, including average (Eq. 4.9), non-biased fully connected layer (last fc) in Eq. 4.10, and target slot only (target): $\mathbf{h}_v^{(L)} = \frac{1}{|\Phi_{target}|} \sum_{t \in \Phi_{target}} \mathbf{h}_v^{(L),t} \in \mathbb{R}^{d_L}$. The results are reported in Table 4.11. Observe that SlotGAT with slot attention is the best on DBLP, IMDB, and ACM, and is top-2 on PubMed_NC and Freebase. The results validate the effectiveness of slot attention. Moreover, SlotGAT (average) and SlotGAT (target) are not with high performance, indicating that blindly averaging slot representations or only using the target slot are less effective. Moreover, We provide the paired t-test results between SlotGAT and its variants *w.r.t.* Micro-F1 and Macro-F1 in Table 4.12 and Table 4.13 respectively. Combining with the results in Table 4.11, we have the following observations. (i) On DBLP, IMDB, ACM and PubMed_NC for Micro-F1 (Table 4.12), the improvement of SlotGAT over SlotGAT(average) and SlotGAT(target slot) is statistically significant, since 7 out of the 8 corresponding p-values are below 0.05, and similar observation can be made on Table 4.13 for these methods. (ii) As shown in Table 4.11, SlotGAT(last fc) is comparable to SlotGAT, and thus the corresponding

Table 4.14: Training time per epoch (millisecond)

	DBLP	IMDB	ACM	PubMed_NC	Freebase
simpleHGN	76.21	75.57	105.44	170.77	423.32
SlotGAT	230.61	241.36	131.98	520.95	458.23
HGT	741.23	613.12	1097.8	2384.14	3711.24

Table 4.15: Inference time (millisecond)

	DBLP	IMDB	ACM	PubMed_NC	Freebase
simpleHGN	57.1	54.42	47.16	70.67	175.34
SlotGAT	127.19	119.95	69.1	258.44	231.31
HGT	531.13	410.52	650.12	1818.42	2939.8

Table 4.16: Peak GPU memory (GB)

	DBLP	IMDB	ACM	PubMed_NC	Freebase
simpleHGN	3.46	3.01	6.32	5.96	7.06
SlotGAT	6.36	5.97	5.5	13.37	9.32
HGT	1.34	1.08	2.02	2.54	6.75

p-values are relatively large in the last rows of Tables 4.12 and 4.13. Note that SlotGAT (last fc) is less explainable than SlotGAT with slot attention. (iii) On Freebase, all methods have close F1 scores in Table 4.11, and thus the p-values are relatively large in the last columns of Tables 4.12 and 4.13. These observations, combined with Table 4.11, validate our statement that SlotGAT with slot attention is the best on DBLP, IMDB, and ACM, and is top-2 on PubMed_NC and Freebase.

Efficiency. Tables 4.14 and 4.15 report the training time and inference time of SlotGAT compared with simpleHGN and HGT, two strong baselines. Table 4.16 reports the peak memory usage. As shown, SlotGAT has moderate training and inference time and slightly higher memory usage to achieve the state-of-the-art effectiveness reported ahead.

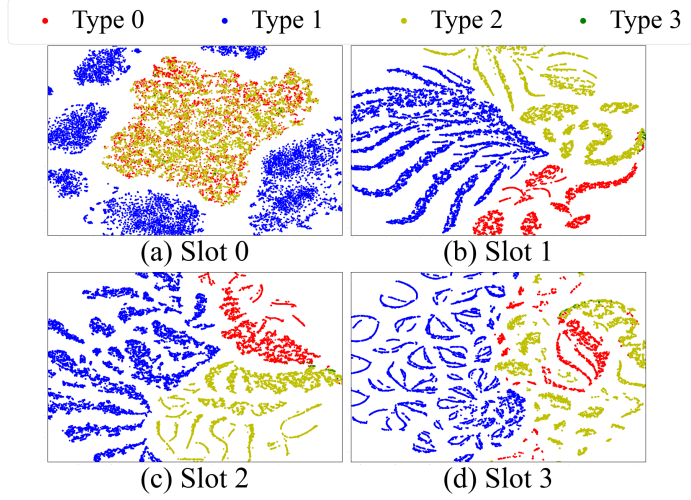


Figure 4.6: T-sne visualization on all slot representations of 3-rd SlotGAT layer on DBLP

4.6 Conclusion, Limitation, and Future Work

In this paper, we identify a semantic mixing issue that potentially hampers the performance on heterogeneous graphs. To alleviate the issue, we design SlotGAT with separate message passing processes in slots, one for each node type. In such a way, SlotGAT conducts slot-based message passing. We also design an attention-based aggregation mechanism over all slot representations to conduct effective slot-wise message aggregation per layer. To effectively support downstream tasks, we further develop a slot attention technique. Extensive experiments validate the superiority of SlotGAT.

SlotGAT has factor $|\Phi|$ as a trade-off of efficiency for effectiveness, which is a potential limitation if many node types exist. There are several ways to handle this in the future, *e.g.*, being selective on node types to reduce the number of slots and setting a limit on the dimension of all slots.

4.6.1 Additional Visualization Results

Figure 4.6 reports the Tsne visualization on the slot representations of the 3-rd SlotGAT layer on DBLP. In particular, Figure 4.6(a) is the visualization of the 0-th slots (in type-0 feature space) of all nodes with colors representing their own types, and Figure 4.5(b), (c), (d) are the visualization of 1-st slots (in type-1 feature space), 2-nd slots (in type-2 feature space), 3-rd slots (in type-3 feature space) of all nodes respectively. First, observe that the 4 slots of a node (for all nodes in a graph) learn very different representations, as the visualizations in the four figures exhibit different patterns. Note that the number of nodes in type 3 is very few compared to other three node types, and thus are nearly invisible in these figures. Therefore, we focus the discussion on nodes of the other three types in red, blue and yellow. In particular, as shown in Figure 4.6(a), the 0-th slots (in type-0 feature space) are able to distinguish nodes in type 1 (blue) from others, but cannot distinguish nodes of types 0, 2, 3. On the other hand, in Figure 4.6(b), the 1-st slots (in type-1 feature space) are able to distinguish nodes of different types in 0, 1, 2. Similar observation can be made in Figures 4.6(c),(d). The visualization in Figure 4.6 proves that SlotGAT is able to learn richer semantics in finer granularity and also preserve the semantic differences of different node-type feature spaces into different slots. This visualization also provide evidence on the superior performance of SlotGAT.

Chapter 5

TINED: GNNs-to-MLPs by Teacher Injection and Dirichlet Energy Distillation

5.1 Introduction

Graph neural networks (GNNs) have delivered impressive outcomes in important applications [48, 66, 121]. The power of GNNs is underpinned by the message passing framework that assimilates and refines node representations by considering their (multi-hop) neighborhood over graphs [138, 162]. Nonetheless, the message passing is computationally demanding due to numerous nodes involved, posing significant challenges to deploy GNNs in latency-sensitive applications that require fast inference [157, 155, 60].

Recent studies attempt to combine the performance advantage of GNNs and the latency advantage of multi-layer perceptrons (MLPs) [55, 159, 116, 18]. Specifically, GLNN [159] distills teacher GNNs into student MLPs via soft labels. Then the student is deployed for fast inference to approximate the performance of GNNs with-

out expensive message passing on graphs. NOSMOG [116] further considers graph structures, robustness, and node relations, while a new graph representation space is learned in [147]. Current research often treats both teacher GNN and student MLP as monolithic model entities, focusing primarily on soft label outputs of the GNN for distillation, which overlooks the intrinsic knowledge contained within fine-grained GNN layers.

In this paper, we conduct an in-depth analysis to reveal important properties of the key operations in GNN layers and propose a novel method, **TINED**, distilling GNNs to MLPs layer-wise by Teacher Injection and Dirichlet Energy Distillation. The main ideas are explained below, and the detailed architecture of **TINED** is presented later.

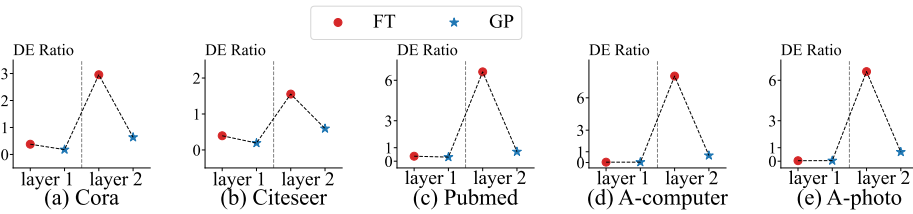


Figure 5.1: The DE ratios of FTs and GPs in the layers of GraphSAGE.

Main Idea of Teacher Injection. A typical GNN layer includes two key operations: feature transformation (FT) and graph propagation (GP) [66, 121, 48, 85, 168]. GP aggregates neighbor representations of a node, while FT transforms node representations with learnable parameters. The valuable knowledge of a teacher GNN is preserved in the well-trained parameters of its FT and GP operations. Existing studies do not directly utilize these parameters [159]. In Section 5.4.1, we recognize that *FT operations in GNNs share the same formulation as fully-connected (FC) layers in MLPs*, both transforming representations using learnable transformation matrices and activations. Thus, we propose to inject the parameters of an FT in a teacher GNN layer into an FC layer in student MLP, followed by fine-tuning for distillation. The GP operation in the GNN layer is emulated by another FC layer in the student. This allows **TINED** to directly transplant teacher knowledge into the student. More-

over, in TINED, the FC layers in the MLP mirror the order of their corresponding FTs and GPs in the GNN teacher, preserving layer-specific knowledge as much as possible. Theoretically, we prove an approximation bound between a GP in GNN and its corresponding FC layer in MLP, which depends on the eigenvalue of graph Laplacian matrix.

Main Idea of Dirichlet Energy Distillation. We further investigate the smoothing properties of FT and GP operations within GNN layers, since it is well-recognized that appropriate smoothing is crucial for GNNs [17]. Importantly, we observe that *the FT and GP operations in a GNN layer often exert opposing smoothing effects: GP aggressively smooths node embeddings, while FT is more restrained and can even diversify embeddings*. Using Dirichlet energy [102], we propose a *DE ratio* measure to quantify whether an operation is conservative (large DE ratio) or aggressive (small DE ratio) in smoothing. Figure 5.1 shows the DE ratios of FTs and GPs in a trained 2-layer GNN teacher GraphSAGE on the experimental datasets. Within the same layer, the DE ratio of FT (red dot) is often larger than that of GP (blue star), indicating their opposing smoothing behaviors. Similar observations are made on other GNNs (see Figure 5.5 and Figure 5.9). To distill these smoothing patterns from GNN to MLP layers, we design Dirichlet Energy Distillation in Section 5.4.3.

We conduct extensive experiments on benchmark datasets under various settings. Results show that TINED achieves superior performance and fast inference speed, compared with existing methods and various teacher GNNs. For example, on the Citeseer data, TINED improves a GNN teacher by 3.94%, MLPs by 15.93%, GLNN by up to 3.21%, and NOSMOG by 1.54%. TINED is $94\times$ faster than its GNN teacher for inference. Our contributions are as follows:

- We propose TINED, a novel method to effectively distill fine-grained layer-wise knowledge from teacher GNNs into student MLPs.
- We develop a teacher injection technique to transplant the parameters of key

operations from GNNs to MLPs. We provide a theoretical approximation analysis.

- We observe distinct smoothing effects of FT and GP operations in GNN layers and introduce Dirichlet energy distillation to impart these effects to student MLPs.
- Extensive experiments demonstrate that TINED achieves superior performance with various GNN teachers on widely-adopted benchmark datasets.

5.2 Related Work

GNNs leverage message-passing to aggregate neighborhood information for learning [66, 121, 48, 21, 69]. GCN [66] introduces layer-wise propagation, GAT [121] employs attention mechanisms, and APPNP [69] uses personalized PageRank. [21] mitigate over-smoothing with residual connections. Extensions include positional encoding [151, 124, 77], few-shot learning [167, 29], and anomaly detection [30, 31, 28]. Specialized GNNs, such as heterogeneous GNNs [166, 156], address specific graph types. Despite their effectiveness, GNN inference is computationally expensive.

Knowledge Distillation has been applied in GNNs to accelerate inference while preserving effectiveness [74, 148, 145, 115, 36]. Previous studies have trained smaller student GNNs with fewer parameters than the large teacher GNNs, such as LSP [148], FreeKD [36] and TinyGNN [141] which still rely on time-intensive message passing. GFKD [27] performs distillation at graph-level instances, while we target nodes within a single graph. Recent studies [116, 147, 159, 55, 161, 135, 136] develop MLP-based student models without message passing. GLNN [159] trains an MLP student using node features as input and soft labels from a GNN teacher as targets. NOSMOG [116] incorporates graph structure, adversarial feature augmentation, and node similarity relations into distillation. VQGraph learns a structure-aware tokenizer to encode

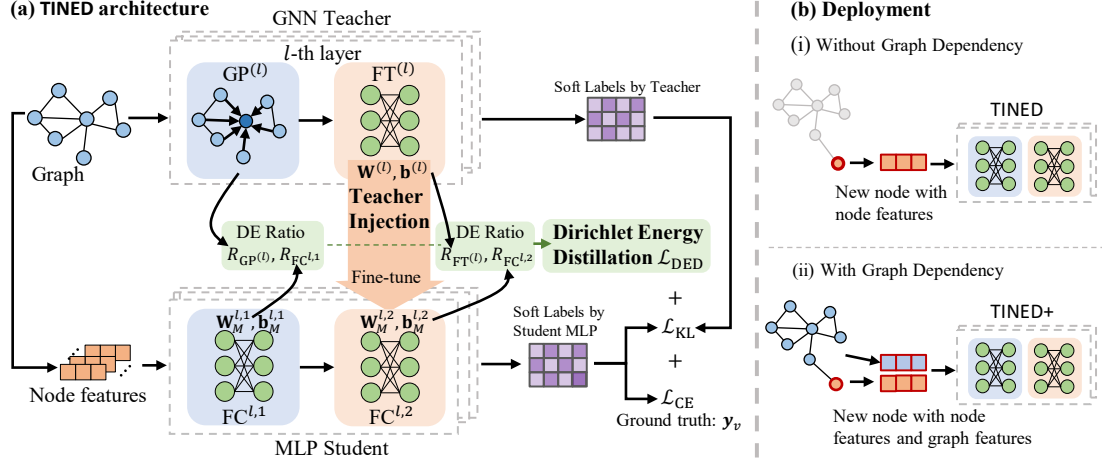


Figure 5.2: (a) TINED with Teacher Injection and Dirichlet Energy Distillation; (b) Inference settings

node substructures [147], during which teacher is also trained. As mentioned, the intrinsic knowledge in well-trained GNN layers is insufficiently utilized by existing studies.

There exist works in other orthogonal domains, also observing the benefits of distilling layer-wise teacher structures, such as language models [81], natural language processing [81, 14, 63] and computer vision [76, 52]. In this work, we fill the gap to develop layer-wise distillation method for GNNs. In addition, [50] work on a different problem that uses pre-trained MLPs for GNN training acceleration.

5.3 Preliminaries

Notations. A graph $\mathcal{G} = (\mathcal{V}, \mathbb{E}, \mathbf{X})$ consists of a node set $\mathcal{V}$ with n nodes, i.e., $|\mathcal{V}| = n$, an edge set $\mathbb{E}$ of size $|\mathbb{E}| = m$, and $\mathbf{X} \in \mathbb{R}^{n \times d_c}$ for d_c -dimensional features of all nodes. Let $\mathbf{A}$ be the adjacency matrix of $\mathcal{G}$, where $\mathbf{A}_{u,v} = 1$ if edge $(u, v) \in \mathbb{E}$, and 0 otherwise. A node v has a neighbor set $\mathcal{N}(v) = \{u | (u, v) \in \mathbb{E}\}$, and the degree of node v is $|\mathcal{N}(v)|$. Degree matrix $\mathbf{D}$ is a diagonal matrix with $\mathbf{D}_{v,v} = |\mathcal{N}(v)|$. In

node classification task, the prediction targets are $\mathbf{Y} \in \mathbb{R}^{n \times K}$, where K is the number of classes, and row $\mathbf{y}_v$ is a K -dim one-hot vector indicating the ground-truth class of node v . A model predicts the class probabilities of node v . In $\mathcal{G}$, we use superscript L to mark the labeled nodes (i.e., $\mathcal{V}^L$, $\mathbf{X}^L$, and $\mathbf{Y}^L$), and superscript U to mark the unlabeled nodes (i.e., $\mathcal{V}^U$, $\mathbf{X}^U$, and $\mathbf{Y}^U$).

GNNs. Let $\mathbf{H}^{(l)} \in \mathbb{R}^{n \times d_l}$ be the output node embedding matrix of the l -th GNN layer, with each row $\mathbf{h}_v^{(l)}$ being the representation of node v in $\mathcal{V}$. Most GNNs fit under the message-passing framework with feature transformation (FT) and graph propagation (GP) operations [168]. The l -th layer usually comprises operations $\text{FT}^{(l)}$ and $\text{GP}^{(l)}$ to get $\mathbf{H}^{(l)}$ in Eq. 5.1. $\text{GP}^{(l)}$ aggregates the $(l-1)$ -th layer representations $\mathbf{H}^{(l-1)}$ over $\mathcal{G}$, and $\text{FT}^{(l)}$ transforms representations to get $\mathbf{H}^{(l)}$. Different GNNs may vary in how they composite $\text{FT}^{(l)}$ and $\text{GP}^{(l)}$.

$$\mathbf{H}^{(l)} = \text{FT}^{(l)} \left(\text{GP}^{(l)} (\mathbf{H}^{(l-1)}, \mathcal{G}) \right) \quad (5.1)$$

MLPs. An MLP is composed of multiple fully-connected (FC) layers. The l -th FC layer converts an embedding $\mathbf{h}^{(l-1)}$ to $\mathbf{h}^{(l)}$ via a transformation matrix $\mathbf{W}_M^{(l)}$ in Eq. 5.2.

$$\mathbf{h}^{(l)} = \sigma(\mathbf{h}^{(l-1)} \mathbf{W}_M^{(l)} + \mathbf{b}^{(l)}), \quad (5.2)$$

where σ is an activation and $\mathbf{b}^{(l)}$ is bias.

GNNs-to-MLPs Distillation. Given a pre-trained GNN teacher, the goal is to train a cost-effective student MLP to predict $\hat{\mathbf{y}}_v$ for node v , utilizing ground-truth labels $\mathbf{y}_v$ of labeled nodes $v \in \mathcal{V}^L$ and soft labels $\mathbf{z}_v$ produced by teacher GNN for all $v \in \mathcal{V}$. The training objective is formulated below [159]. Note that the student is trained by both ground truth and soft labels from the teacher.

$$\mathcal{L} = \sum_{v \in \mathcal{V}^L} \mathcal{L}_{CE}(\hat{\mathbf{y}}_v, \mathbf{y}_v) + \lambda \sum_{v \in \mathcal{V}} \mathcal{L}_{KL}(\hat{\mathbf{y}}_v, \mathbf{z}_v), \quad (5.3)$$

where $\mathcal{L}_{CE}$ is the cross-entropy loss by comparing the student MLP predictions $\hat{\mathbf{y}}_v$ with the ground truth $\mathbf{y}_v$, and $\mathcal{L}_{KL}$ is the KL-divergence loss between the student

MLP predictions $\hat{\mathbf{y}}_v$ and the soft labels $\mathbf{z}_v$ from teacher, and weight λ balances the two losses.

5.4 The TINED Method

In Eq. 5.3, existing studies primarily use GNN soft labels and ground truth labels for distillation. Our approach TINED expands this to include model parameters and layer-specific properties of teacher GNNs for distilling knowledge into MLPs, as illustrated in Figure 5.2. As shown in Figure 5.2(a), for the Teacher Injection technique, we identify that the $\text{FT}^{(l)}$ operation of the l -th GNN layer essentially conducts the same computation as an FC layer in an MLP, and thus, we inject the parameters of $\text{FT}^{(l)}$ into an FC layer $\text{FC}^{l,2}$ of the student MLP. The injected parameters are fine-tuned during the distillation to ensure controlled adaption. To emulate the $\text{GP}^{(l)}$ operation in the l -th GNN layer, we employ another FC layer $\text{FC}^{l,1}$ in the student MLP. We establish a theoretical approximation bound between $\text{GP}^{(l)}$ and $\text{FC}^{l,1}$. Then as in Figure 5.2(a), we use Dirichlet energy to develop DE ratios $R_{\text{GP}^{(l)}}$ and $R_{\text{FT}^{(l)}}$ to quantify the smoothing effects of $\text{GP}^{(l)}$ and $\text{FT}^{(l)}$ in the l -th GNN layer, and design Dirichlet Energy Distillation with a $\mathcal{L}_{DED}$ loss to ensure the DE ratios $R_{\text{FC}^{l,2}}$ and $R_{\text{FC}^{l,1}}$ of the student MLP’s FC layers $\text{FC}^{l,1}$ and $\text{FC}^{l,2}$ preserve the smoothing effects of $\text{GP}^{(l)}$ and $\text{FT}^{(l)}$ from the teacher GNN.

There are two typical inference deployment settings as depicted in Figure 5.2(b). TINED can operate without graph dependency for inference [159], which is useful when the graph structure is unavailable or when new unseen nodes lack graph connections; TINED can also work with graph structures when allowed [116]. Following [159, 116], we use GraphSAGE [48] with GCN aggregation as the teacher GNN to explain TINED. Our experiments include results over different teacher GNNs. In Section 5.5.7, we explain how to use TINED for various GNNs.

5.4.1 Teacher Injection

The idea is to inject the parameters of certain operations of GNNs into MLPs, thereby directly transferring the teacher’s knowledge to the student. The GNN operations to be injected should be compatible with MLP in Eq. 5.2, i.e., with the same formulation. As shown in Eq. 5.1, a GNN layer consists of GP and FT operations. In what follows, we show that the FT operations of GNNs can be injected into student MLPs. We use GraphSAGE as an example, while the analysis on other GNNs as teacher is in Section 5.5.7. The l -th layer of GraphSAGE with $\text{GP}^{(l)}$ and $\text{FT}^{(l)}$ operations is

$$\begin{aligned} \text{GP}^{(l)}: \tilde{\mathbf{h}}_v^{(l)} &= \| (\mathbf{h}_v^{(l-1)}, \text{AGG}^{(l)}(\{\mathbf{h}_u^{(l-1)}, \forall u \in \mathcal{N}(v)\})) , \\ \text{FT}^{(l)}: \mathbf{h}_v^{(l)} &= \sigma(\tilde{\mathbf{h}}_v^{(l)} \cdot \mathbf{W}^{(l)} + \mathbf{b}^{(l)}) . \end{aligned} \quad (5.4)$$

In Eq. 5.4, $\text{GP}^{(l)}$ operation has an aggregator $\text{AGG}^{(l)}$ to combine the $(l-1)$ -th representations $\mathbf{h}_u^{(l-1)}$ of v ’s neighbors and then concatenates ($\|$) them with $\mathbf{h}_v^{(l-1)}$ to get $\tilde{\mathbf{h}}_v^{(l)}$. In $\text{FT}^{(l)}$ operation, a learnable transformation matrix $\mathbf{W}^{(l)}$ is applied to $\tilde{\mathbf{h}}_v^{(l)}$, followed by an activation function to yield the l -th layer representation $\mathbf{h}_v^{(l)}$ of v .

Observe that (i) $\text{FT}^{(l)}$ operates independently of the graph, whereas $\text{GP}^{(l)}$ requires the graph; (ii) $\text{FT}^{(l)}$ in Eq. 5.4 has the same formulation as an FC layer of an MLP in Eq. 5.2. Therefore, to approximate the l -th GNN layer, we employ two FC layers $\text{FC}^{l,1}$ and $\text{FC}^{l,2}$ in the student MLP. In Eq. 5.5, the $\text{FC}^{l,1}$ layer of the student MLP is to approximate $\text{GP}^{(l)}$ in Eq. 5.4, while the $\text{FC}^{l,2}$ layer approximates $\text{FT}^{(l)}$ in Eq. 5.4, with teacher injection to set $\mathbf{W}_M^{l,2} = \mathbf{W}^{(l)}$ and $\mathbf{b}_M^{l,2} = \mathbf{b}^{(l)}$, transferring the parameters of $\text{FT}^{(l)}$ to $\text{FC}^{l,2}$. $\mathbf{s}_v^{(l)}$ is the embedding of node v generated by the student.

$$\begin{aligned} \text{FC}^{l,1} \text{ for } \text{GP}^{(l)}: \hat{\mathbf{s}}_v^{(l)} &= \sigma(\mathbf{s}_v' \mathbf{W}_M^{l,1} + \mathbf{b}_M^{l,1}) , \\ \text{FC}^{l,2} \text{ for } \text{FT}^{(l)}: \mathbf{s}_v^{(l)} &= \sigma(\hat{\mathbf{s}}_v \mathbf{W}_M^{l,2} + \mathbf{b}_M^{l,2}) , \end{aligned} \quad (5.5)$$

where $\mathbf{W}_M^{l,2} = \mathbf{W}^{(l)}$ and $\mathbf{b}_M^{l,2} = \mathbf{b}^{(l)}$, and $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are from Eq. 5.4.

Different from existing methods, TINED distills knowledge from GNNs to MLPs on a per-layer basis. Directly injecting GNN parameters into the MLP is anticipated

to improve the effectiveness. For layer-wise GNNs with T layers, the resulting MLP will have $2T$ FC layers. Typically, T is small, e.g., 2 or 3, and the efficiency impact on inference between MLPs with 2 and 4 layers is minimal (1.45ms vs. 1.63ms, as reported in [159], compared to 153ms by GraphSAGE). Furthermore, our experiments show that TINED can achieve a favorable tradeoff between effectiveness and efficiency for inference. For decoupled GNNs, e.g., APPNP [69], with FTs and GPs decoupled, the teacher injection is also applicable with a small number of FC layers (see Section 5.5.7).

In Theorem 2, we provide a theoretical approximation bound between $\text{GP}^{(l)}$ in the l -th layer of teacher GNN and its corresponding $\text{FC}^{l,1}$ in student MLP, serving as an attempt to establish the relationship between GNNs and MLPs for knowledge distillation. The proof is in Section 5.4.2.

Theorem 2. *For a sparse matrix $\mathbf{L} \in \mathbb{R}^{n \times n}$ and a feature matrix $\mathbf{H} \in \mathbb{R}^{n \times d}$ with $\text{rank}(\mathbf{H}) = d$, there exists a transformation matrix $\mathbf{W}^*$ to approximate $\mathbf{LH}$ by $\mathbf{HW}^*$ with relative error*

$$\frac{\|\mathbf{LH} - \mathbf{HW}^*\|_F}{\|\mathbf{H}\|_F} \leq \lambda_{\max}(\mathbf{L}),$$

where $\|\cdot\|_F$ is the Frobenius norm and $\lambda_{\max}(\mathbf{L})$ is the largest eigenvalue of laplacian $\mathbf{L}$.

5.4.2 Proof of Theorem 2

Proof. We denote the $\mathbf{H}^\dagger$ as the pseudo-inverse of matrix $\mathbf{H}$. Since $\text{rank}(\mathbf{H}) = d$, $\mathbf{H}$ has linearly independent columns, its pseudo-inverse could be represented as $\mathbf{H}^\dagger = (\mathbf{H}^\top \mathbf{H})^{-1} \mathbf{H}^\top$. Moreover, given the SVD factorization of matrix $\mathbf{H} = \mathbf{U} \begin{pmatrix} \mathbf{S} \\ 0 \end{pmatrix} \mathbf{V}^\top$, where $\mathbf{U} \in \mathbb{R}^{N \times N}$ and $\mathbf{V} \in \mathbb{R}^{d \times d}$ are unitary, and $\mathbf{S} \in \mathbb{R}^{d \times d}$ is invertible diagonal matrix, we have $\mathbf{H}^\dagger = \mathbf{V} \begin{pmatrix} \mathbf{S}^{-1} & 0 \end{pmatrix} \mathbf{U}^\top$.

Denote $\tilde{\mathbf{H}} = \mathbf{L}\mathbf{H}$, then we have

$$\|\tilde{\mathbf{H}} - \mathbf{H}\mathbf{W}\|_F^2 = \sum_{i=1}^d \|\tilde{\mathbf{H}}_{:,i} - \mathbf{H}\mathbf{W}_{:,i}\|_2^2 \quad (5.6)$$

Let $\mathbf{W}^* = \mathbf{H}^\dagger \tilde{\mathbf{H}}$, we have $\mathbf{W}_{:,i}^* = \mathbf{H}^\dagger \tilde{\mathbf{H}}_{:,i}$. Then

$$\begin{aligned} \mathbf{H}\mathbf{W}_{:,i}^* &= \mathbf{H}\mathbf{H}^\dagger \tilde{\mathbf{H}}_{:,i} \\ &= \mathbf{U} \begin{pmatrix} I_d & 0 \\ 0 & 0_{n-d} \end{pmatrix} \mathbf{U}^\top \tilde{\mathbf{H}}_{:,i} \end{aligned} \quad (5.7)$$

Then

$$\begin{aligned} \|\tilde{\mathbf{H}}_{:,i} - \mathbf{H}\mathbf{W}_{:,i}^*\|_2 &= \|\tilde{\mathbf{H}}_{:,i} - \mathbf{H}\mathbf{H}^\dagger \tilde{\mathbf{H}}_{:,i}\|_2 \\ &\leq \|I_n - \mathbf{H}\mathbf{H}^\dagger\|_2 \|\tilde{\mathbf{H}}_{:,i}\|_2 \\ &\leq \|\tilde{\mathbf{H}}_{:,i}\|_2 \\ &= \|\mathbf{L}\mathbf{H}_{:,i}\|_2 \\ &\leq \|\mathbf{L}\|_2 \|\mathbf{H}_{:,i}\|_2 \\ &= \lambda_{\max}(\mathbf{L}) \|\mathbf{H}_{:,i}\|_2 \end{aligned} \quad (5.8)$$

Thus we have

$$\begin{aligned} \|\tilde{\mathbf{H}} - \mathbf{H}\mathbf{W}\|_F^2 &= \sum_{i=1}^d \|\tilde{\mathbf{H}}_{:,i} - \mathbf{H}\mathbf{W}_{:,i}\|_2^2 \\ &\leq \lambda_{\max}^2(\mathbf{L}) \sum_{i=1}^d \|\mathbf{H}_{:,i}\|_2^2 \\ &= \lambda_{\max}^2(\mathbf{L}) \|\mathbf{H}\|_F^2. \end{aligned} \quad (5.9)$$

Finally we get

$$\frac{\|\mathbf{L}\mathbf{H} - \mathbf{H}\mathbf{W}^*\|_F}{\|\mathbf{H}\|_F} \leq \lambda_{\max}(\mathbf{L}) \quad (5.10)$$

□

Specifically, Laplacian matrix $\mathbf{L}$ represents graph topology used in $\text{GP}^{(l)}$, $\mathbf{L}\mathbf{H}$ represents the output of $\text{GP}^{(l)}$, while $\mathbf{H}\mathbf{W}^*$ represents the output of $\text{FC}^{l,1}$ and $\mathbf{H}\mathbf{W}^*$ approximates $\mathbf{L}\mathbf{H}$. The relative error is upper-bounded by the largest eigenvalue $\lambda_{\max}(\mathbf{L})$ of $\mathbf{L}$, regardless of graph size.

The parameters of FC layers $\text{FC}^{l,2}$ in Eq. 5.5 are sourced from the $\text{FT}^{(l)}$ operations of teacher GNN. To ensure these FC layers contribute to the student MLP’s training in a regulated manner, we introduce a gradient modifier η to fine-tune the injected parameters when updating the gradients,

$$\begin{aligned}\hat{\nabla} \mathbf{W}_M^{l,2} &= \eta \nabla \mathbf{W}_M^{l,2} \\ \hat{\nabla} \mathbf{b}_M^{l,2} &= \eta \nabla \mathbf{b}_M^{l,2},\end{aligned}\tag{5.11}$$

where η is a hyperparameter to control fine-tuning, while ∇ represents gradient.

In other words, during training, the update of the parameters in $\text{FC}^{l,2}$ based on gradient optimization after backward propagation is adjusted by η parameter.

5.4.3 Dirichlet Energy Distillation

It is widely recognized that an appropriate degree of smoothing is crucial for the efficacy of GNNs. Here we make an interesting observation that, within the l -th GNN layers, $\text{GP}^{(l)}$ operations tend to aggressively smooth node embeddings, while $\text{FT}^{(l)}$ operations often apply conservative smoothing or even diversify embeddings. These distinct layer-level smoothing behaviors in teacher GNNs should be captured in the student MLPs.

To achieve this, we introduce Dirichlet Energy Distillation. Dirichlet energy [102] is a measure commonly used to quantify the degree of smoothing in embeddings by evaluating node pair distances. Definition 2 defines the Dirichlet energy $E(\mathbf{H})$ of a node embedding matrix $\mathbf{H}$.

Definition 2. *Given a node embedding matrix $\mathbf{H} \in \mathbb{R}^{n \times d}$, learned from either the GNN teacher or the MLP student at a certain layer, the Dirichlet energy of $\mathbf{H}$ is*

$$E(\mathbf{H}) = \frac{1}{n} \cdot \text{tr}(\mathbf{H}^\top \mathbf{L} \mathbf{H}),\tag{5.12}$$

where $\text{tr}(\cdot)$ is the trace of a matrix, and $\mathbf{L} = \mathbf{D} - \mathbf{A}$ is the Laplacian matrix of graph $\mathcal{G}$.

A lower Dirichlet energy value $E(\mathbf{H})$ suggests that the embeddings in $\mathbf{H}$ are smooth, whereas a higher value indicates diversity among the embeddings. For an operation op that processes $\mathbf{H}$ to output $op(\mathbf{H})$, if $E(op(\mathbf{H})) < E(\mathbf{H})$, then the operation op is smoothing the embeddings; on the other hand, if $E(op(\mathbf{H})) > E(\mathbf{H})$, op diversifies them.

We define Dirichlet energy ratio R_{op} (DE ratio) as follows.

Definition 3 (DE ratio). *The DE ratio R_{op} of an operation op is the Dirichlet energy of its output $E(op(\mathbf{H}))$ over the Dirichlet energy of its input $E(\mathbf{H})$, $R_{op} = \frac{E(op(\mathbf{H}))}{E(\mathbf{H})}$.*

On a trained teacher GraphSAGE with 2 layers, in each layer, we calculate the DE ratios $R_{\text{FT}^{(l)}}$ and $R_{\text{GP}^{(l)}}$ of $\text{FT}^{(l)}$ and $\text{GP}^{(l)}$ operations respectively. We repeat 10 times and report the average DE ratios in Figure 5.1, over the benchmark datasets used in experiments.

While the overall Dirichlet energy of embeddings is decreasing in GraphSAGE, in a specific l -th layer shown in Figure 5.1, we make the following two consistent observations about DE ratio. (i) Within the same layer for $l = 1, 2$, the DE ratio $R_{\text{FT}^{(l)}}$ of $\text{FT}^{(l)}$ consistently exceeds the DE ratio $R_{\text{GP}^{(l)}}$ of $\text{GP}^{(l)}$, demonstrating that $\text{GP}^{(l)}$ operation actively smooths embeddings, whereas $\text{FT}^{(l)}$ operation is relatively conservative for smoothing. (ii) At $l = 2$, DE ratio $R_{\text{FT}^{(l)}}$ even surpasses 1, indicating that in this layer, $\text{FT}^{(l)}$ acts to diversify embeddings rather than smoothing them. These trends are consistently observed across layer-wise GNNs, including GCN and GAT in Figure 5.5 and Figure 5.9.

Figure 5.1 reveals that FTs and GPs can have opposing effects on smoothing. Recall that we associate $\text{FT}^{(l)}$ and $\text{GP}^{(l)}$ of the l -th GNN layer with $\text{FC}^{l,2}$ and $\text{FC}^{l,1}$ layers in the student MLP in Eq. 5.5. The proposed Dirichlet Energy Distillation technique aims to encapsulate the distinct smoothing behaviors of $\text{FT}^{(l)}$ and $\text{GP}^{(l)}$ into $\text{FC}^{l,2}$ and $\text{FC}^{l,1}$ respectively, thereby transferring teacher GNN’s knowledge of smoothing effects to student MLP.

Specifically, the loss of the l -th GNN layer for Dirichlet Energy Distillation, $\mathcal{L}_{DED}^{(l)}$, is the sum of the squares of the difference between DE ratios $R_{\text{GP}^{(l)}}$ of $\text{GP}^{(l)}$ and $R_{\text{FC}^{l,1}}$ of the $\text{FC}^{l,1}$ layer, and the difference between $R_{\text{FT}^{(l)}}$ of $\text{FT}^{(l)}$ and $R_{\text{FC}^{l,2}}$ of the $\text{FC}^{l,2}$ layer. The total DED loss $\mathcal{L}_{DED}$ is the sum of $\mathcal{L}_{DED}^{(l)}$ for all GNN layers $l = 1, \dots, T$.

$$\begin{aligned}\mathcal{L}_{DED}^{(l)} &= (R_{\text{GP}^{(l)}} - R_{\text{FC}^{l,1}})^2 + (R_{\text{FT}^{(l)}} - R_{\text{FC}^{l,2}})^2 \\ \mathcal{L}_{DED} &= \sum_{l=1}^T \mathcal{L}_{DED}^{(l)}.\end{aligned}\tag{5.13}$$

The final objective function $\mathcal{L}$ of our method TINED is the weighted combination of ground truth cross-entropy loss $\mathcal{L}_{CE}$, soft label distillation loss $\mathcal{L}_{KL}$, and Dirichlet Energy Distillation loss $\mathcal{L}_{DED}$:

$$\mathcal{L} = \sum_{v \in \mathcal{V}^L} \mathcal{L}_{CE}(\hat{\mathbf{y}}_v, \mathbf{y}_v) + \lambda \sum_{v \in \mathcal{V}} \mathcal{L}_{KL}(\hat{\mathbf{y}}_v, \mathbf{z}_v) + \beta \sum_{l=1}^T \mathcal{L}_{DED}^{(l)},\tag{5.14}$$

where λ and β are weights for balancing the loss functions.

5.4.4 Discussion on Why to Choose Dirichlet Energy

We choose Dirichlet Energy as the distillation criterion because it is a principled measure for quantifying the global smoothness of graph signals[102], effectively capturing the fundamental smoothing and diversifying behaviors inherent to GNN layers. Unlike alternative regularization methods such as cosine similarity or feature matching, which focus on local, point-wise consistency and have shown only marginal improvements in prior studies like NOSMOG[116], Dirichlet Energy provides a holistic view of the graph structure. By aligning these global statistics, our method ensures the student MLP faithfully reproduces the topological dynamics of the teacher GNN.

5.5 Experiments

5.5.1 Datasets

In Table 5.1, we provide the statistics of the datasets in experiments. For all datasets, we follow the setting in [159, 145] to split the data. Specifically, for the first five datasets, we use the splitting in [145] and each random seed corresponds to a different split. For the OGB datasets Arxiv and Products, we follow the OGB official splits based on time and popularity respectively.

Table 5.1: Dataset Statistics and Splits.

Dataset	# Nodes	# Edges	# Features	# Classes	# Train	# Val	# Test
Cora	2,485	5,069	1,433	7	140	210	2135
Citeseer	2,110	3,668	3,703	6	120	180	1810
Pubmed	19,717	44,324	500	3	60	90	19567
A-computer	13,381	245,778	767	10	200	300	12881
A-photo	7,487	119,043	745	8	160	240	7087
Arxiv	169,343	1,166,243	128	40	90941	29799	48603
Products	2,449,029	61,859,140	100	47	196615	39323	2213091

5.5.2 Experiment Settings

Datasets. We use 7 widely used public benchmark datasets, including Cora, Citeseer, Pubmed, A-computer, A-photo [159, 145], and Arxiv and Products [54] that are two large OGB datasets, to evaluate our method and baselines. Table 5.1 in Section 5.5.1 provides the data statistics and splits.

Teacher Architectures. Following [159, 116], for the main results, GraphSAGE [48] with GCN aggregation is used as the teacher model. We also conduct experiments of different GNN teachers, e.g., GCN, GAT and APPNP in Section 5.5.7. The details of teachers and hyperparameter settings are in Section 5.5.3.

Methods. TINED does not require graph dependency for inference, similar to GLNN [159]. When graph dependency is permissible, we enhance it to create TINED+, akin to

Table 5.2: Under the transductive(*tran*) setting, the accuracy results of the teacher and methods without graph dependency for inference on $\mathcal{V}^U$ (higher is better). The best is in bold. OOM means out-of-memory.

Datasets	Teacher	Without Graph Dependency					
	SAGE	MLP	FFG2M	KRD	GLNN	GLNN*	TINED
Cora	80.64±1.57	59.18±1.60	82.38±1.41	82.27±1.31	80.26±1.66	81.31±1.62	82.63±1.57
Citeseer	70.49±1.53	58.50±1.86	72.85±1.59	72.84±1.70	71.22±1.50	72.38±1.40	74.43±1.53
Pubmed	75.56±2.06	68.39±3.09	76.56±3.41	77.01±3.11	75.59±2.46	76.95±2.72	77.09±2.14
A-computer	82.82±1.37	67.62±2.21	83.67±1.04	82.87±0.87	82.71±1.18	83.64±1.13	85.18±1.12
A-photo	90.85±0.87	77.29±1.79	93.18±0.87	92.82±0.74	91.95±1.04	92.99±0.63	93.97±0.53
Arxiv	70.73±0.35	55.67±0.24	58.51±0.35	59.26±0.51	63.75±0.48	63.78±0.69	64.44±0.72
Products	77.17±0.32	60.02±0.10	OOM	OOM	63.71±0.31	65.56±0.26	69.48±0.25

Table 5.3: Under the transductive(*tran*) setting, the accuracy results of the teacher and methods with graph dependency for inference on $\mathcal{V}^U$ (higher is better). The best is in bold.

Datasets	Teacher	With Graph Dependency		
	SAGE	NOSMOG	NOSMOG*	TINED+
Cora	80.64±1.57	83.04±1.26	82.27±1.75	83.70±1.02
Citeseer	70.49±1.53	73.78±1.54	73.85±2.27	75.39±1.59
Pubmed	75.56±2.06	77.34±2.36	76.79±2.65	77.75±3.14
A-computer	82.82±1.37	84.04±1.01	84.33±1.14	84.82±1.58
A-photo	90.85±0.87	93.36±0.69	93.57±0.48	94.05±0.39
Arxiv	70.73±0.35	71.65±0.29	71.17±0.60	71.52±0.34
Products	77.17±0.32	78.45±0.38	78.47±0.28	78.59±0.28

NOSMOG [116], to incorporate graph structural information. Baselines GLNN and NOSMOG use their original layer and hidden dimension configurations as per their papers. Additionally, we configure them with the same layer and hidden dimension settings as ours, dubbed as GLNN* and NOSMOG*, for evaluation. We further compare with KRD and FFG2M [136, 135], as well as with MLPs and the GraphSAGE teacher.

Transductive and Inductive Settings. Two settings are considered [159, 116]: transductive (*tran*) setting, and production (*prod*) setting with both inductive and transductive evaluations (*ind* & *tran*).

tran: A model is trained on $\mathcal{G}$, $\mathbf{X}$, and $\mathbf{Y}^L$, and soft labels $\mathbf{z}_v$ of all nodes in $\mathcal{V}$ are used for knowledge distillation. Inference evaluation is conducted on the nodes in $\mathcal{V}^U$.

prod (*ind* & *tran*): we randomly select out 20% of nodes from the unlabeled nodes $\mathcal{V}^U$, dividing $\mathcal{V}^U$ into disjoint inductive (unobserved) subset and observed subset, $\mathcal{V}^U = \mathcal{V}_{obs}^U \sqcup \mathcal{V}_{ind}^U$. Node features and labels are partitioned into disjoint sets, i.e. $\mathbf{X} = \mathbf{X}^L \sqcup \mathbf{X}_{obs}^U \sqcup \mathbf{X}_{ind}^U$, and $\mathbf{Y} = \mathbf{Y}^L \sqcup \mathbf{Y}_{obs}^U \sqcup \mathbf{Y}_{ind}^U$. Let $\mathcal{G}_{obs}$ be the graph induced from $\mathcal{G}$, with edges connecting nodes in $\mathcal{V}^L \sqcup \mathcal{V}_{obs}^U$. A model is trained on $\mathcal{G}_{obs}$, $\mathbf{X}^L$, $\mathbf{X}_{obs}^U$, and $\mathbf{Y}^L$. Soft labels in subsets $\mathcal{V}^L \sqcup \mathcal{V}_{obs}^U$ are used for distillation. In this *prod* (*ind* & *tran*) setting, inference is evaluated on $\mathcal{V}_{ind}^U$ for *ind* and $\mathcal{V}_{obs}^U$ for *tran* (which is different from the *tran* setting over $\mathcal{V}^U$ explained above). The overall *prod* performance is the weighted sum of *ind* performance on $\mathcal{V}_{ind}^U$ and *tran* on $\mathcal{V}_{obs}^U$, with weights proportional to the size of $\mathcal{V}_{ind}^U$ and $\mathcal{V}_{obs}^U$.

Evaluation. Following [116, 159], we present the mean and standard deviation of performance results from 10 trials, each with a unique random seed. A method is evaluated by accuracy with the best model chosen on validation data and tested on test data.

Computer Resource Details

The experiments on both baselines and our approach are implemented using PyTorch,

the DGL [128] library for GNN algorithms, and Adam [65] for optimization. We run all experiments on Intel(R) Xeon(R) Platinum 8338C CPU @ 2.60GHz CPU, and a Nvidia Geforce 3090 Cards with Cuda version 11.7. Source codes of all competitors are obtained from respective authors. The totally training time cost for one set of hyper-parameters varies among datasets, from Cora costing 10 minutes to Products costing 13 hours.

Table 5.4: Hyperparameters for GNNs on five datasets from [159, 116].

Dataset	GraphSAGE	GCN	GAT
# layers	2	2	2
hidden dim	128	64	64
learning rate	0.01	0.01	0.01
weight decay	0.0005	0.001	0.01
dropout	0	0.8	0.6
fan out	5,5	-	-
attention heads	-	-	8

5.5.3 Teacher details and hyperparameter search space

The hyperparameters of GNN models on each dataset are taken from the best hyperparameters provided by previous studies [116, 159]. For APPNP teacher model, we found that the suggested hyper-parameter in previous literature [159, 116] produces relatively poor results, thus we search it in following space, achieving a better APPNP teacher: # layers from [2, 3], learning rate from [0.0001, 0.001, 0.01], weight decay from [0.0001, 0.001, 0.01], dropout from [0, 0.5, 0.8], hidden dim from [128, 256], power iteration K from [5, 10, 15]. For the students MLP, GLNN and NOSMOG, we set the number of layers and the hidden dimension of each layer to be the same as the teacher GNN, so their total number of parameters stays the same as the teacher GNN. For TINED, TINED+, GLNN* and NOSMOG*, given a layer-wise GNN teacher with T layers (including, GraphSAGE, GCN, and GAT), in the student MLP we set the

same hidden dimension as teachers, and $2T$ layers, while for APPNP, we set the student MLP to have $T_1 + 1$ layers as explained in Section 5.5.7. Other hyper-parameter searching spaces are listed here: Learning rate from $[0.0001, 0.0005, 0.001, 0.005, 0.01]$, weight decay from $[0.0, 0.0001, 0.0005, 0.001, 0.005, 0.01]$, weight of distillation λ from $[0.1, 0.4, 0.5, 0.6, 1]$, normamlization type from [batch normalization, layer normalization, none], dropout from $[0, 0.1, 0.3, 0.5, 0.8]$. Batch size for two large OGB datasets from $[512, 1024, 4096]$. Weight of DED β from $[1e^{-6}, 5e^{-5}, 1e^{-5}, 0.05, 0.1, 0.5, 1, 5, 10]$. Fine tuning weight η for injected teacher FT layers from $[0.01, 0.1, 0.5, 1, 3, 10]$. For the hyperparamrter space of TINED+ from NOSMOG, the search space is the same as [116]. In large OGB datasets, the direct computation of DE values will run out of GPU memory 24GB. Thus, we propose a sampling ratio ζ to compute approximate DE values in DED process. This is done by inducing a subgraph based on random sampling of edges, then approximate the DE ratio of input and output feature matrix on this sampled subgraph. We also search this sampling proportion ζ from $[0.001, 0.005, 0.1, 0.4, 1]$. Moreover, empirically we found that applying smoothing term on the DE ratio in MSE loss could benefit optimization, thus we propose a smoothing function technique μ on DE ratio R_{op} , i.e. $\mu(R_{op})$, when computing the DED loss, where μ is searched from $[\text{sqrt}(\cdot), \log(\cdot), \text{Identity}(\cdot)]$ [59].

5.5.4 Performance in Transductive Setting

Table 5.3 and Table 5.2 report the results under the transductive setting with inference on node set $\mathcal{V}^U$. Table 5.3 and Table 5.2 can be directly comparable to those reported in literature [116, 159, 54, 145]. Table 5.3 and Table 5.2, GNNs-to-MLPs methods are in two categories: with or without graph dependency for inference. Baselines GLNN and NOSMOG use their original student MLP configurations for layers and hidden dimensions as recommended in their respective papers. GLNN* and NOSMOG* (with an asterisk *) adopt the same configurations as our model for a fair comparison. TINED surpasses all baselines without graph dependency, often

with a significant margin. For example, on the large OGB Products dataset, TINED achieves gains of 9.42%, 5.77%, and 3.92% over MLP, GLNN, and GLNN*, while baselines KRD and FFG2M are OOM. On Citeseer, TINED achieves 74.43% accuracy, improving by 15.93%, 3.21%, 2.05%, 1.59%, and 1.58% over MLP, GLNN, GLNN*, KRD, and FFG2M. The performance of TINED validates the power of the proposed techniques to preserve layer-level knowledge into MLPs. Similar to GLNN, TINED surpasses the teacher on the first five data, while being effective on the last two. When graph dependency is allowed, TINED+ excels NOSMOG and NOSMOG* on most datasets, except Arxiv where the performance is comparable. On Citeseer, TINED+ achieves 75.39% accuracy, 1.54% higher than NOSMOG*. Moreover, all methods with graph dependency, including TINED+ and NOSMOG, perform better than those without graph, and TINED+ excels the teacher model on all datasets, indicating the extensibility of TINED+ to consider graph structures for inference.

5.5.5 Performance in Production Setting with *ind* & *tran*

We then conduct experiments under the *prod* (*ind* & *tran*) setting with results in Tables 5.5 and 5.6. Note that the *tran* results in Tables 5.5 and 5.6 are over $\mathcal{V}_{obs}^U$, different from the results in Table 5.3 and Table 5.2 on $\mathcal{V}^U$ in Section 5.5.4. In Tables 5.5 and 5.6, TINED and TINED+ can outperform the teacher model and the baseline methods in categories with or without graph dependency under almost all settings. For instance, without graph dependency for inference, TINED is better than baselines, except Arxiv and Pubmed where close performance is achieved. As an example, on A-computer under *ind* setting, TINED achieves 82.83% accuracy, significantly improving GLNN* by 1.73%. With graph dependency, TINED+ surpasses NOSMOG methods on all datasets except *prod* and *tran* setting on Cora, where the performance is comparable. For instance, on Citeseer under *prod*, TINED+ achieves performance gain of 2.98% and 2.25% over NOSMOG and NOSMOG* respectively. Moreover, on Arxiv and Products with a significant distribution shift between training and test data [159],

Table 5.5: In *prod* (*ind* & *tran*) setting, the accuracy results of the teacher and all methods without graph dependency for online inference are presented. *ind* indicates the results on $\mathcal{V}_{ind}^U$, *tran* indicates the results on $\mathcal{V}_{obs}^U$, and *prod* indicates the weighted sum of the performance of both *ind* and *tran* with weights proportional to the size of $\mathcal{V}_{ind}^U$ and $\mathcal{V}_{obs}^U$. The best result in each category is in bold. The teacher’s performance surpassing the best result in at least one category is italicized.

		Teacher	Without Graph Dependency					
Datasets	Eval	SAGE	MLP	FFG2M	KRD	GLNN	GLNN*	TINED
Cora	<i>prod</i>	79.53	59.18	78.60	75.74	77.82	78.14	78.90
	<i>ind</i>	<i>81.03±1.71</i>	59.44±3.36	72.02±1.43	70.26±1.94	73.21±1.50	73.58±1.42	74.38±1.28
	<i>tran</i>	79.16±1.60	59.12±1.49	80.01±1.41	77.11±1.44	78.97±1.56	79.65±1.45	80.04±1.50
Citeseer	<i>prod</i>	68.06	58.49	71.89	71.38	69.08	70.91	72.29
	<i>ind</i>	69.14±2.99	59.31±4.56	69.75±3.16	69.78±3.04	68.48±2.38	71.10±1.50	72.68±1.97
	<i>tran</i>	67.79±2.80	58.29±1.94	72.12±2.69	71.77±2.81	69.23±2.39	70.86±1.66	72.20±1.66
Pubmed	<i>prod</i>	74.77	68.39	73.98	76.00	74.67	75.21	75.79
	<i>ind</i>	75.07±2.89	68.28±3.25	73.49±7.91	75.17±3.11	74.52±2.95	74.83±2.83	75.64±3.02
	<i>tran</i>	74.70±2.33	68.42±3.06	74.10±7.78	76.20±3.00	74.70±2.75	75.30±2.70	75.83±2.81
A-computer	<i>prod</i>	82.73	67.62	82.69	81.17	82.10	83.23	84.46
	<i>ind</i>	<i>82.83±1.51</i>	67.69±2.20	80.52±1.56	79.15±1.82	80.27±2.11	81.10±1.49	82.83±1.45
	<i>tran</i>	82.70±1.34	67.60±2.23	83.23±1.36	81.67±1.92	82.56±1.80	83.77±1.36	84.87±1.38
A-photo	<i>prod</i>	90.45	77.29	92.35	91.84	91.34	91.54	93.38
	<i>ind</i>	90.56±1.47	77.44±1.50	90.70±0.76	90.04±1.12	89.50±1.12	89.35±0.89	91.96±0.72
	<i>tran</i>	90.42±0.68	77.25±1.90	92.77±0.24	92.29±0.63	91.80±0.49	92.09±0.71	93.74±0.51
Arxiv	<i>prod</i>	<i>70.69</i>	55.35	59.60	59.32	63.50	64.17	63.24
	<i>ind</i>	<i>70.69±0.58</i>	55.29±0.63	57.02±0.43	57.32±0.31	59.04±0.46	58.73±0.46	59.79±0.46
	<i>tran</i>	<i>70.69±0.39</i>	55.36±0.34	60.24±0.23	59.82±0.27	64.61±0.15	65.53±0.30	64.10±0.38
Products	<i>prod</i>	<i>76.93</i>	60.02	OOM	OOM	63.47	68.48	69.35
	<i>ind</i>	<i>77.23±0.24</i>	60.02±0.09	OOM	OOM	63.38±0.33	68.13±0.20	68.68±0.27
	<i>tran</i>	<i>76.86±0.27</i>	60.02±0.11	OOM	OOM	63.49±0.31	68.57±0.20	69.52±0.27

Table 5.6: In *prod* (*ind* & *tran*) setting, the accuracy results of the teacher and all methods with graph dependency for online inference are presented. *ind* indicates the results on $\mathcal{V}_{ind}^U$, *tran* indicates the results on $\mathcal{V}_{obs}^U$, and *prod* indicates the weighted sum of the performance of both *ind* and *tran* with weights proportional to the size of $\mathcal{V}_{ind}^U$ and $\mathcal{V}_{obs}^U$. The best result in each category is in bold. The teacher’s performance surpassing the best result in at least one category is italicized.

		Teacher	With Graph Dependency		
Datasets	Eval	SAGE	NOSMOG	NOSMOG*	TINED+
Cora	<i>prod</i>	79.53	81.02	80.30	80.77
	<i>ind</i>	<i>81.03±1.71</i>	81.36±1.53	80.98±2.39	81.50±2.54
	<i>tran</i>	79.16±1.60	80.93±1.65	80.13±1.64	80.59±1.45
Citeseer	<i>prod</i>	68.06	70.60	71.33	73.58
	<i>ind</i>	69.14±2.99	70.30±2.30	72.35±2.99	74.20±1.67
	<i>tran</i>	67.79±2.80	70.67±2.25	71.07±1.85	73.43±1.63
Pubmed	<i>prod</i>	74.77	75.82	75.57	75.90
	<i>ind</i>	75.07±2.89	75.87±3.32	75.49±2.96	76.30±2.95
	<i>tran</i>	74.70±2.33	75.80±3.06	75.58±2.83	75.80±2.88
A-computer	<i>prod</i>	82.73	83.85	85.02	85.08
	<i>ind</i>	<i>82.83±1.51</i>	84.36±1.57	85.23±1.51	85.45±1.60
	<i>tran</i>	82.70±1.34	83.72±1.44	84.97±1.44	84.98±1.32
A-photo	<i>prod</i>	90.45	92.47	92.00	93.12
	<i>ind</i>	90.56±1.47	92.61±1.09	92.93±0.96	93.27±0.85
	<i>tran</i>	90.42±0.68	92.44±0.51	91.77±0.69	93.08±0.68
Arxiv	<i>prod</i>	<i>70.69</i>	70.90	70.95	71.22
	<i>ind</i>	<i>70.69±0.58</i>	70.09±0.55	70.12±0.39	70.42±0.35
	<i>tran</i>	<i>70.69±0.39</i>	71.10±0.34	71.16±0.15	71.43±0.19
Products	<i>prod</i>	<i>76.93</i>	77.33	78.25	78.91
	<i>ind</i>	<i>77.23±0.24</i>	77.02±0.19	78.52±0.22	79.31±0.29
	<i>tran</i>	<i>76.86±0.27</i>	77.41±0.21	78.18±0.23	78.81±0.29

TINED+ outperforms the teacher and NOSMOG, showing the capability of our techniques on large real-world graph datasets. We conclude that TINED and TINED+ can achieve excellent performance in *prod* setting with *ind* & *tran*.

5.5.6 Inference Time

Figure 5.3 shows the trade-off between accuracy and inference time on Citeseer. Methods closer to the top-left corner achieve a better balance of accuracy and speed. Our methods, TINED and TINED+, achieve the highest accuracy (74.43% and 75.39%) while being fast (1.63ms and 1.64ms). In contrast, GNNs are slower, e.g., 2-layer GraphSAGE (SAGE-L2) takes 153.14ms, and 3-layer GraphSAGE (SAGE-L3) takes 1202.45ms. TINED and TINED+ are 94 times faster than SAGE-L2 and 733 times faster than SAGE-L3. All the distillation methods (NOSMOG, GLNN*, TINED, TINED+, KRD and FFG2M) have similar efficiency in around 1-2ms, with negligible differences. Compared with SAGE teacher that needs 153.14ms, the efficiency of distillation techniques in our design is validated.

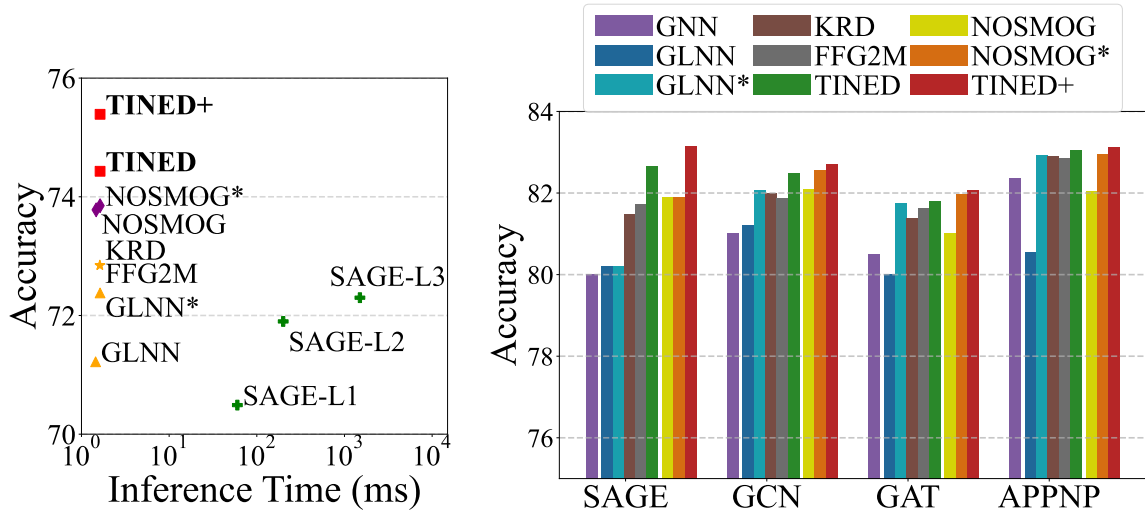


Figure 5.3: Inference Time and Figure 5.4: Accuracy on Different Teacher GNNs Accuracy

5.5.7 Different Teacher GNN Architectures

Figure 5.4, we show that TINED and TINED+ can maintain strong performance with alternative GNN teachers, including GCN, GAT, and APPNP, in addition to GraphSAGE. We report the average performance of a distillation method with different teacher GNNs across the five benchmark datasets. With different teachers, TINED can always learn more effective student MLPs than GLNN, GLNN*, KRD, and FFG2M. With graph dependency, TINED+ can maintain its strong performance with different teachers and surpasses NOSMOG and NOSMOG*. When different teacher GNNs are adopted, Figure 5.4 illustrates the effectiveness of our methods to preserve layer-level knowledge into MLPs.

Implementing Other GNNs as Teacher

Here we explain how to implement other GNNs as teacher in TINED, especially on how to apply teacher injection in The Proposed Model section to different GNNs.

GCN. The message passing of GCN in the l -th layer is

$$\begin{aligned} \text{GP}^{(l)}: \tilde{\mathbf{h}}_v^{(l)} &= \hat{\mathbf{L}}\mathbf{H}^{(l-1)}, \\ \text{FT}^{(l)}: \mathbf{h}_v^{(l)} &= \sigma\left(\tilde{\mathbf{h}}_v^{(l)} \cdot \mathbf{W}^{(l)} + \mathbf{b}^{(l)}\right). \end{aligned} \quad (5.15)$$

where $\hat{\mathbf{L}} = \hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}$ is the normalized Laplacian of the graph, $\hat{\mathbf{A}} = \mathbf{A} + I_n$ is the adjacency matrix with self loops, $\hat{\mathbf{D}}_{ii} = \sum_j \hat{\mathbf{A}}_{ij}$ is the degree matrix, $\mathbf{W}^{(l)}$ is the transformation matrix of l -th layer, and $\mathbf{H}^{(l)}$ is the output of l -th layer.

For the l -th layer of the GCN teacher, we directly inject $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ of the teacher in Equation 5.15 into the student FC layer $\text{FC}^{l,2}$ in Equation 5.5, i.e., $\mathbf{W}_M^{l,2} = \mathbf{W}^{(l)}$ and $\mathbf{b}_M^{l,2} = \mathbf{b}^{(l)}$, while using $\text{FC}^{l,1}$ in Equation 5.5 to simulate $\text{GP}^{(l)}$ in Equation 5.15.

GAT. The message passing of GAT in the l -th layer with single head attention is

$$\begin{aligned} \text{GP}^{(l)}: \tilde{\mathbf{h}}_v^{(l)} &= \Pi^{(l)}\mathbf{H}^{(l-1)}, \\ \text{FT}^{(l)}: \mathbf{h}_v^{(l)} &= \sigma\left(\tilde{\mathbf{h}}_v^{(l)} \cdot \mathbf{W}^{(l)} + \mathbf{b}^{(l)}\right). \end{aligned} \quad (5.16)$$

where $\Pi^{(l)}$ is the attention matrix defined as $\Pi_{ij}^{(l)} = \text{LeakyReLU}([\mathbf{H}_i^{(l-1)}\mathbf{W}^{(l)} \parallel \mathbf{H}_j^{(l-1)}\mathbf{W}^{(l)}]^\top \mathbf{a}^{(l)})$ if (i, j) is an edge in $\mathcal{G}$, i.e., $(i, j) \in \mathcal{E}$, and $\Pi_{ij}^{(l)} = 0$ otherwise, $\parallel$ denotes the concatenation operation, and $\mathbf{a}^{(l)} \in \mathbb{R}^{2d_l}$ is a learnable attention vector.

For the l -th layer of the GAT teacher, we directly inject $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ of the teacher in Equation 5.16 into the student, i.e., $\mathbf{W}_M^{l,2} = \mathbf{W}^{(l)}$ and $\mathbf{b}_M^{l,2} = \mathbf{b}^{(l)}$, while using $\text{FC}^{l,1}$ in Equation 5.5 to simulate $\text{GP}^{(l)}$ in Equation 5.16.

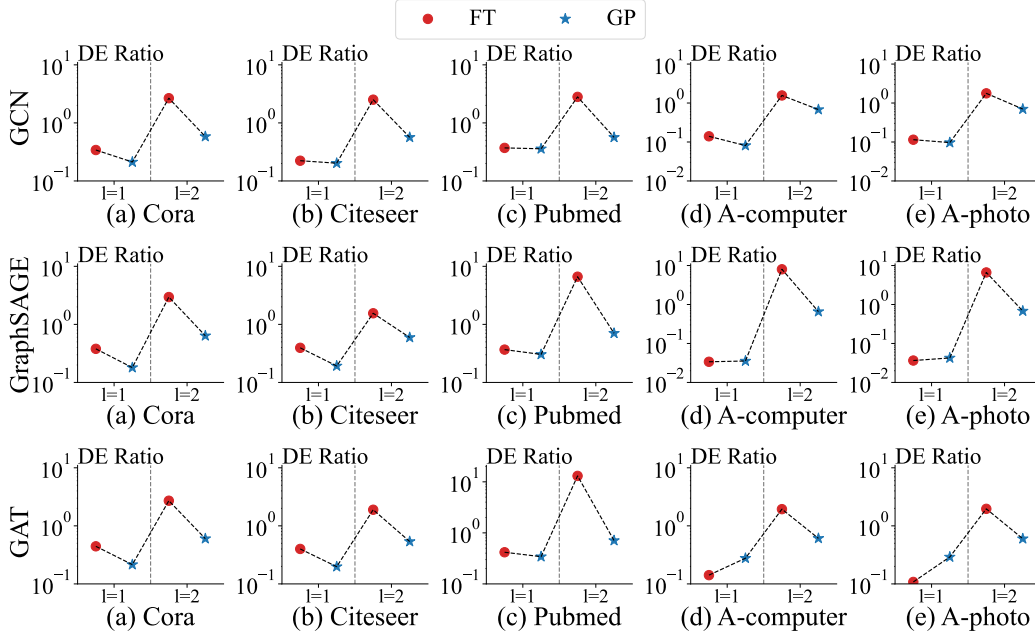


Figure 5.5: The layer-wise DE ratio of trained GCN, GraphSAGE and GAT, with 2 layers architecture on different datasets. The x -axis is the layer number, and the y -axis is the DE ratio value.

APPNP. APPNP decouples feature transformation and graph propagation into two stages. APPNP first performs FTs for T_1 times in Equation 5.17 to get $\mathbf{H}^{(T_1)}$ (usually $T_1 = 2$), and then propagates $\mathbf{H}^{(T_1)}$ over the graph for T_2 hops in Equation 5.18 by approximate Personalized PageRank.

$$\text{FT}^{(l)}: \mathbf{H}^{(l)} = \mathbf{H}^{(l-1)}\mathbf{W}^{(l)}, l = 1, 2, \dots, T_1, \quad (5.17)$$

$$\text{GP}: \mathbf{H}^{(l)} = (1 - \alpha)\hat{\mathbf{L}}\mathbf{H}^{(l)} + \alpha\mathbf{H}^{(T_1)}, l = T_1 + 1, T_1 + 2, \dots, T_1 + T_2, \quad (5.18)$$

where α is a hyperparameter and $\hat{\mathbf{L}}$ is the normalized Laplacian mentioned before.

For each of $\text{FT}^{(l)}$ in APPNP, we inject it into a FC^l layer in the student MLP. In other words, we have T_1 FC layers to mimic the FT layers in APPNP. Then for the T_2 steps of propagation in Equation 5.18, we use one FC layer FC^{T_1+1} to mimic all of them, since T_2 can be large. In total, there are $T_1 + 1$ FC layers in MLP to approximate the teacher GNN. The DED loss of APPNP is $\mathcal{L}_{DED} = \sum_{l=1}^{T_1} (R_{\text{FT}^{(l)}} - R_{\text{FC}^l})^2 + (R_{\text{GP}} - R_{\text{FC}^{T_1+1}})^2$

5.5.8 Experimental Analysis

Ablation Study. Denote TINED-TIN as TINED without Teacher Injection (TIN), and TINED-DED as TINED without Dirichlet Energy Distillation (DED). The results are in Table 5.7. Both TIN and DED contribute positively, and TINED achieves the best performance on all datasets, demonstrating the effectiveness of our techniques in TINED.

Table 5.7: Ablation study

Datasets	TINED-TIN	TINED-DED	TINED	Δ_{TIN}	Δ_{DED}
Cora	81.32±1.60	81.64±1.64	82.63±1.57	↑ 1.31%	↑ 0.99%
Citeseer	71.88±1.27	74.12±1.42	74.43±1.53	↑ 2.55%	↑ 0.31%
Pubmed	75.93±3.40	76.45±2.68	77.09±2.14	↑ 1.16%	↑ 0.64%
A-computer	84.59±1.78	84.80±1.67	85.18±1.12	↑ 0.59%	↑ 0.38%
A-photo	93.90±0.52	93.67±0.69	93.97±0.53	↑ 0.07%	↑ 0.30%
Arxiv	63.72±0.89	64.39±0.80	64.44±0.72	↑ 0.72%	↑ 0.05%
Products	69.20±0.25	68.63±0.26	69.48±0.25	↑ 0.28%	↑ 0.85%

Visualization. Based on the visualization of teacher SAGE’s output embeddings after 500 epochs of training (Figure 5.6(a)), we compare the embeddings of TINED and GLNN* both at initialization (epoch 0) and after training convergence (epoch

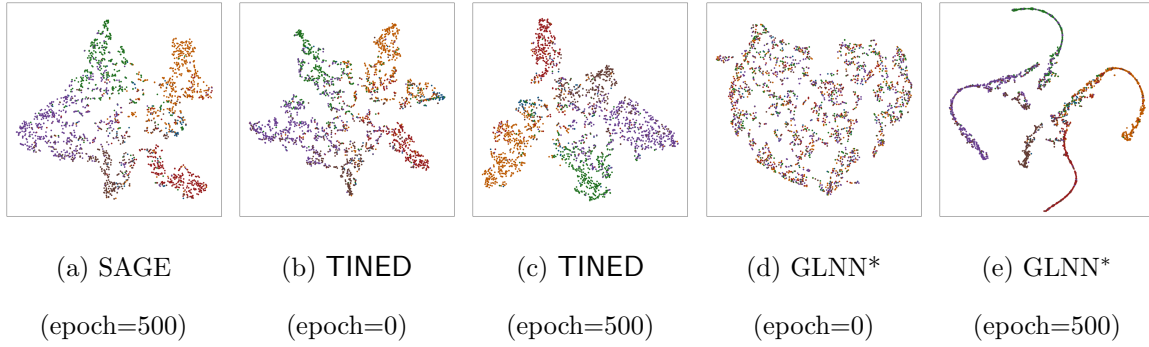


Figure 5.6: t-SNE of model embeddings at different training stages on Citeseer. (a) SAGE (teacher model at epoch 500), (b, c) TINED (epochs 0 and 500), (d, e) GLNN* (epochs 0 and 500).

500). Figure 5.6(b) shows TINED initialized (epoch 0) using the proposed teacher injection technique, resulting in an embedding visualization similar to that of the teacher in Figure 5.6(a). In contrast, the visualization of GLNN* at epoch 0 appears quite random, as seen in Figure 5.6(d). Upon reaching training convergence (epoch 500), TINED in Figure 5.6(c) maintains a visualization closely resembling the teacher’s in Figure 5.6(a), whereas the baseline GLNN* displays a noticeably different visualization. This evaluation confirms the effectiveness of the proposed teacher injection and Dirichlet energy distillation techniques employed by TINED. More visualization results are in Figure 5.7.

5.5.9 Layer-wise t-SNE Visualization

In Figure 5.7, we present a detailed layer-wise t-SNE visualization of the embeddings for various models and stages. Figure 5.7(a) (1st row) shows the 2-layer Graph-SAGE teacher model. Figure 5.7(b) (2nd row) illustrates TINED at initialization before training (epoch 0), while Figure 5.7(c) (3rd row) displays TINED after training convergence (epoch 500). Figure 5.7(d) (4th row) and Figure 5.7(e) (5th row) depict the baseline GLNN* at initialization (epoch 0) and after training convergence

(epoch 500), respectively. For the 2-layer GraphSAGE in Figure 5.7(a), the 1st row

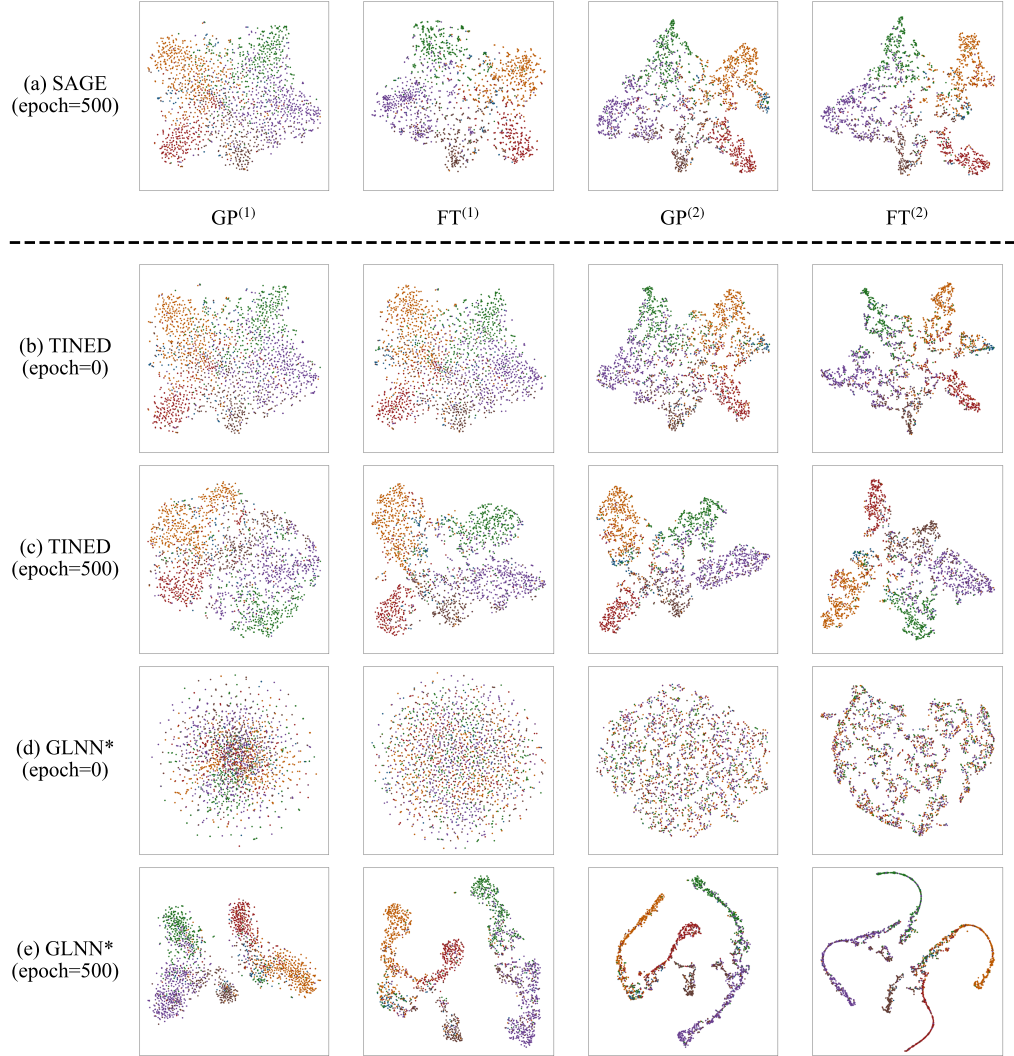


Figure 5.7: Layer-wise t-SNE visualization of the embeddings for various models (GraphSAGE, TINED, GLNN*) and stages (epoch 0 at initialization and epoch 500 after training convergence) on Citeseer.

visualizes the embeddings from left to right: after the $FT^{(1)}$ operation in the first layer, the $GP^{(1)}$ operation in the first layer, $FT^{(2)}$ in the second layer, and $GP^{(2)}$ in the second layer. For each distillation method in Figures 5.7(b-e), each row contains four plots representing the embeddings of the four fully-connected (FC) layers in the methods, corresponding to the four operations in the first row of the teacher. Com-

pared to the teacher model in Figure 5.7(a), TINED at initialization in Figure 5.7(b) already exhibits quite similar embedding patterns, thanks to the proposed teacher injection technique. In contrast, the baseline GLNN* shows a random visual pattern at epoch 0 in Figure 5.7(d), which is radically different from the teacher model. After training convergence at 500 epochs, TINED’s visualized embeddings in Figure 5.7(c) closely resemble those of the teacher in Figure 5.7(a), whereas the baseline GLNN* in Figure 5.7(e) produces embeddings that are visually distinct. These observations demonstrate the effectiveness of the teacher injection and Dirichlet energy distillation techniques in TINED.

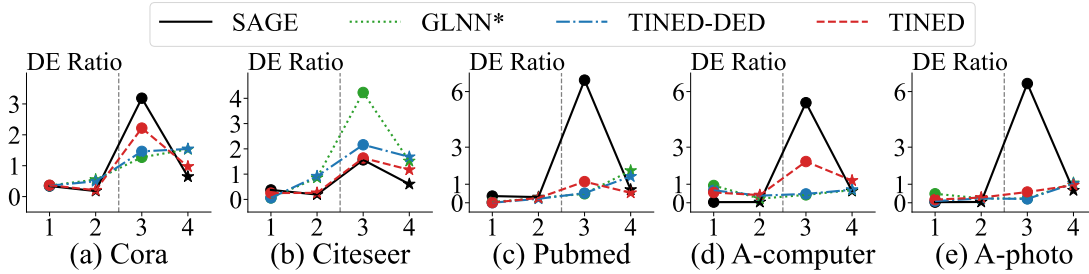


Figure 5.8: Learned DE ratios in MLPs v.s. ground truth of 2-layer GraphSAGE.

DE ratio of TINED on GraphSAGE teacher Recall that Figure 5.1 shows the ground-truth DE ratios of teacher GraphSAGE, and we propose Dirichlet Energy Distillation in Section 5.4.3 to preserve smoothing effects into MLPs. Figure 5.8 reports the learned DE ratios of TINED, TINED-DED, and GLNN*. TINED is more closely aligned to the ground-truth DE ratios, highlighting that TINED effectively retains the GNN smoothing properties into MLPs. **DE Ratios of other GNN architectures**

To show the pattern of DE ratios in different GNN teachers, we plot the layer-wise DE ratios of GCN, GraphSAGE and GAT, in addition to GraphSAGE, with 2 layers architecture on the five benchmarks in Figure 5.5. From Figure 5.5 we can observe that all GNNs share very similar patterns across datasets: within a specific layer, in Figure 5.5, we observe the following nearly consistent patterns about *DE-ratio*: (i)

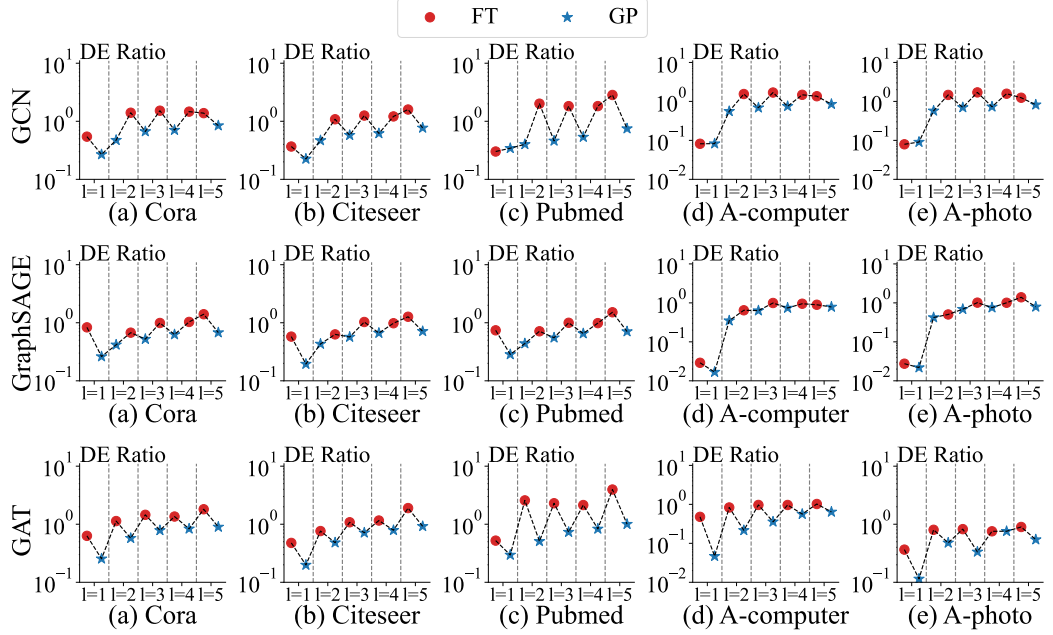


Figure 5.9: The layer-wise DE ratio of trained GCN, GraphSAGE and GAT, with 5 layers architecture on different datasets. The x -axis is the layer number, and the y -axis is the DE ratio value.

within the same layer for $l = 1, 2$, most DE ratio $R_{\text{FT}^{(l)}}$ for $\text{FT}^{(l)}$ is larger than $R_{\text{GP}^{(l)}}$ of $\text{GP}^{(l)}$, suggesting that $\text{GP}^{(l)}$ actively smooths embeddings, whereas $\text{FT}^{(l)}$ is relatively conservative for smoothing; (ii) at $l = 2$, DE ratio $R_{\text{FT}^{(l)}}$ even surpasses 1, indicating that in this layer, $\text{FT}^{(l)}$ acts to diversify embeddings rather than smoothing them. Furthermore, to investigate the smoothing behavior on deeper layer cases, we plot Figure 5.9 showing layer-wise DE ratios of 5-layer trained GNNs, and from Figure 5.9 we can observe that the trained 5-layer GNNs have similar patterns to the 2-layer GNNs: the DE ratios of $\text{FT}^{(l)}$ is generally higher than that of $\text{GP}^{(l)}$, indicating that $\text{GP}^{(l)}$ is aggressive while $\text{FT}^{(l)}$ is relatively conservative for smoothing.

Approximation Bound. Theorem 2 shows a bound between $\text{GP}^{(l)}$ in GNNs and its counterpart $\text{FC}^{l,1}$ in MLPs. Table 5.8 reports the errors of TINED and GLNN* using $\text{FC}^{l,1}$ to approximate $\text{GP}^{(l)}$, $\|\text{GP}^{(l)}(\mathbf{H}) - \text{FC}^{l,1}(\mathbf{H})\|_F / \|\mathbf{H}\|_F$, for $l = 1, 2$. When $l = 2$, the errors of TINED are lower than GLNN*, while the errors for $l = 1$ are comparable,

Table 5.8: Approximation error

	$l = 1$		$l = 2$	
Datasets	TINED	GLNN*	TINED	GLNN*
Cora	1.32	1.64	0.90	1.12
Citeseer	0.46	0.86	0.26	0.75
Pubmed	0.88	0.82	0.74	1.02
A-computer	1.89	1.88	2.55	2.65
A-photo	1.21	0.83	1.40	1.54

showing TINED effectively approximates $GP^{(l)}$.

Table 5.9: Vary β of TINED on transductive setting. Best results are in bold.

β	Cora	Citeseer	Pubmed	A-computer	A-photo
1e-09	81.33±1.49	73.39±1.28	76.63±2.42	84.70±1.15	93.41±0.61
1e-06	81.64±1.71	73.39±1.30	76.28±2.76	85.17±1.21	93.48±0.65
0.001	81.64±1.57	73.39±1.31	76.51±2.64	84.80±1.21	93.65±0.68
0.1	81.71±1.59	73.65±1.38	77.10±2.15	84.70±0.99	93.97±0.58
1.0	82.61±1.58	73.81±1.29	75.84±2.66	84.29±1.08	86.59±5.66
10.0	80.92±2.15	74.57±1.42	71.57±2.76	71.35±6.30	78.00±3.19

Parameter Sensitivity Analysis. In our method, the parameter η controls the degree of fine-tuning in Eq. 5.11, while β controls the importance of Dirichlet Energy Distillation in Eq. 5.14. Tables 5.10 and 5.9 present the accuracy results of varying η and β of TINED across multiple datasets. In Table 5.10, as η increases, TINED demonstrates a clear trend where performance initially improves and then declines, with the best results highlighted in bold. A similar pattern is observed for β in Table 5.9, emphasizing the trade-off controlled by these parameters. Note that hyperparameter tuning is essential in machine learning research. Section 5.5.3 of the paper details the search space for our parameters.

The parameter ζ controls the subgraph sampling ratio used to estimate the DE ratio,

Table 5.10: Vary η of TINED on transductive setting. Best results are in bold.

η	Cora	Citeseer	Pubmed	A-computer	A-photo
1e-09	73.80 $\pm$ 2.56	67.20 $\pm$ 1.86	73.39 $\pm$ 2.10	72.53 $\pm$ 2.17	86.06 $\pm$ 2.38
1e-06	73.50 $\pm$ 2.48	67.19 $\pm$ 1.87	73.32 $\pm$ 1.88	72.73 $\pm$ 1.99	86.35 $\pm$ 1.79
0.001	77.12 $\pm$ 0.91	67.24 $\pm$ 1.83	73.69 $\pm$ 1.83	84.13 $\pm$ 1.23	87.30 $\pm$ 2.08
0.01	79.26 $\pm$ 1.30	69.38 $\pm$ 1.70	75.12 $\pm$ 2.32	85.17$\pm$1.21	90.48 $\pm$ 1.38
0.1	81.40 $\pm$ 1.69	71.52 $\pm$ 1.58	76.55 $\pm$ 2.81	83.61 $\pm$ 1.67	93.97$\pm$0.54
0.5	82.01 $\pm$ 1.64	74.57$\pm$1.42	77.10$\pm$2.15	83.23 $\pm$ 1.29	93.51 $\pm$ 0.60
1.0	82.61$\pm$1.58	73.57 $\pm$ 1.39	76.65 $\pm$ 2.77	82.85 $\pm$ 0.91	93.37 $\pm$ 0.51
10.0	78.85 $\pm$ 1.63	73.56 $\pm$ 1.48	75.61 $\pm$ 2.73	70.63 $\pm$ 4.87	89.53 $\pm$ 0.78

helping to avoid memory overflow on large datasets. We vary ζ and report the results on the large ogbn-arxiv dataset in Table 5.11. The results show that TINED maintains stable performance across different ζ values. Notably, ζ values larger than 0.8 are unnecessary, as they increase computational overhead without improving distillation quality.

Table 5.11: Vary ζ for TINED on large graph ogbn-arxiv under transductive setting.

ζ	0.1	0.3	0.5	0.7	>0.8
Arxiv	64.31 $\pm$ 0.71	64.32 $\pm$ 0.91	64.38 $\pm$ 0.75	64.44 $\pm$ 0.80	OOM

Heterophilic Datasets We conducted experiments on representative heterophilic datasets, Squirrel and Amazon-ratings, under the production setting. Squirrel [101] is a web page dataset collected from Wikipedia, while Amazon-ratings [97] is a product co-purchasing network based on data from SNAP Datasets. In Table 5.13 and Table 5.12, our methods, TINED and TINED+, consistently outperform existing approaches across all settings, often by a significant margin. For instance, on the Squirrel dataset without graph dependency, TINED achieves a *prod* performance of 41.95%, surpassing the best competitor GLNN, which achieves 39.90%. Similarly, on the Amazon-ratings

Table 5.12: Experiment results under *prod* (*ind* & *tran*) setting on heterophilic datasets - Methods without graph dependency. The best result in each category is in bold.

Datasets	Eval	Teacher	Without Graph Dependency				
		SAGE	FFG2M	KRD	GLNN	GLNN*	TINED
Squirrel	<i>prod</i>	35.47	38.34	37.16	39.90	39.70	41.95
	<i>ind</i>	41.44±4.66	42.00±4.78	42.11±4.77	44.89±5.67	45.00±4.56	46.89±5.23
	<i>tran</i>	33.98±1.66	37.43±3.34	35.93±3.36	38.65±1.09	38.37±1.02	40.72±1.36
Amazon-ratings	<i>prod</i>	47.55	50.33	49.56	49.87	49.41	50.70
	<i>ind</i>	47.45±1.48	47.55±0.97	47.68±1.00	47.72±1.00	47.71±1.14	49.02±1.02
	<i>tran</i>	47.58±0.48	51.03±1.73	50.03±1.50	50.41±0.45	49.84±0.39	51.12±0.44

Table 5.13: Experiment results under *prod* (*ind* & *tran*) setting on heterophilic datasets - Methods with graph dependency. The best result in each category is in bold.

Datasets	Eval	Teacher	With Graph Dependency		
		SAGE	NOSMOG	NOSMOG*	TINED+
Squirrel	<i>prod</i>	35.47	38.17	39.43	40.89
	<i>ind</i>	41.44±4.66	45.44±4.75	44.33±3.24	46.78±3.80
	<i>tran</i>	33.98±1.66	36.35±1.69	38.20±1.59	39.42±1.05
Amazon-ratings	<i>prod</i>	47.55	47.86	48.80	50.42
	<i>ind</i>	47.45±1.48	47.47±1.45	48.46±1.37	49.51±1.54
	<i>tran</i>	47.58±0.48	47.96±0.31	48.88±0.56	50.65±0.56

dataset with graph dependency, TINED+ achieves a *prod* accuracy of 50.42%, representing a 1.62% improvement over the best competitor NOSMOG*. These results highlight the effectiveness of our proposed methods in handling heterophilic datasets.

Comparison with Vanilla MLP with Additional Layers. In addition to the vanilla MLP with 2 layers, we also evaluate a 4-layer MLP, denoted as MLP*. Tables 5.14 and 5.15 present the results under both transductive and production settings. Our method, TINED, consistently outperforms MLP* across all datasets. Interestingly, MLP* demonstrates degraded performance compared to the 2-layer MLP, suggesting that additional layers lead to overfitting on these datasets. For instance, on the Pubmed dataset, MLP* achieves 67.61%, which is inferior to the 69.41% achieved by the 2-layer MLP. In contrast, TINED significantly improves upon MLP*, achieving a notable accuracy of 77.09%, highlighting its robustness and effectiveness.

Table 5.14: Compare with MLP* with 4 layers under transductive setting.

	Cora	Citeseer	Pubmed	A-computer	A-photo
MLP	60.84±1.08	63.41±1.96	69.41±2.88	70.07±1.77	80.19±1.48
MLP*	58.20±3.16	60.81±3.48	67.61±1.51	66.84±3.03	78.66±2.48
TINED	82.63±1.57	74.43±1.53	77.09±2.14	85.18±1.12	93.97±0.53

Table 5.15: Comparison with MLP* with 4 layers under *prod(ind&tran)* setting

	Eval	Cora	Citeseer	Pubmed	A-computer	A-photo
MLP	<i>ind</i>	61.31±2.16	63.95±2.95	69.66±2.68	70.36±2.48	79.76±2.00
	<i>tran</i>	60.88±1.41	62.99±2.39	69.67±2.61	69.92±2.03	79.53±2.05
MLP*	<i>ind</i>	58.67±2.42	62.15±3.50	67.76±1.88	68.09±2.60	77.27±2.17
	<i>tran</i>	58.12±1.81	61.46±2.46	68.07±1.87	67.90±2.41	77.05±2.48
TINED	<i>ind</i>	74.38±1.28	72.68±1.97	75.64±3.02	82.83±1.45	91.96±0.72
	<i>tran</i>	80.04±1.50	72.20±1.66	75.83±2.81	84.87±1.38	93.74±0.51

5.6 Conclusion

We introduce TINED, a novel method for distilling GNN knowledge into MLPs. Our approach includes Teacher Injection, which directly transfers well-trained parameters from GNNs to MLPs, and Dirichlet Energy Distillation, which preserves the unique smoothing effects of key GNN operations within MLPs. Comprehensive experiments demonstrate that TINED outperforms existing methods across various settings and seven datasets. Currently, the number of layers in the student MLP is dependent on the number of layers in the GNN teacher. In future work, we aim to develop new techniques to accelerate the process and allow for a flexible number of fully connected layers by considering the intrinsic properties of GNNs and MLPs. Moreover, we will explore the potential of TINED in other types of graphs, including heterogeneous graphs and dynamic graphs, to further enhance its applicability and effectiveness.

Chapter 6

Conclusion

6.1 Summary of Contributions

This thesis has addressed three fundamental challenges that limit the practical applicability of Graph Neural Networks (GNNs): data scarcity, structural complexity, and computational efficiency. Through systematic investigation and novel methodological developments, we have made significant contributions to each of these areas.

6.1.1 Effective Learning with Limited Labels

In Chapter 3, we identified and addressed the critical issue of training instability in GNNs when working with extremely sparse labeled data. Our analysis revealed that existing GNN methods suffer from high variance in prediction distributions when only one labeled node per class is available, leading to unreliable performance across different training runs.

Our proposed Stabilized Self-Training framework introduces two key innovations. First, we developed a stabilized pseudo-labeling technique that maintains class distribution balance and selects high-confidence predictions as reliable pseudo-labels.

Second, we implemented a negative sampling regularization approach that further improves classification accuracy by effectively leveraging unlabeled data.

The experimental results demonstrate substantial improvements over baseline methods. For instance, our framework improved GCN performance by 17.9% and DAGNN by 6.6% on the challenging one-label-per-class setting. These results show that proper handling of pseudo-label quality and training stability can significantly enhance GNN performance in data-scarce scenarios.

6.1.2 Advancing Heterogeneous Graph Modeling

Chapter 4 tackled the semantic mixing problem that occurs when heterogeneous GNNs aggregate information from different node types. Traditional approaches force transformations from different type-specific feature spaces into unified spaces, leading to information loss and reduced model expressiveness.

Our SlotGAT architecture introduces a novel slot-based message passing mechanism that maintains separate semantic spaces for different node types throughout the learning process. Each node maintains multiple slots corresponding to different node types, enabling the preservation of type-specific semantics while allowing effective information propagation across the heterogeneous graph structure.

The slot-based approach, combined with attention mechanisms for adaptive importance weighting, achieved superior performance across all evaluated datasets. This work demonstrates that preserving semantic diversity in heterogeneous graphs is crucial for effective representation learning and downstream task performance.

6.1.3 Efficient GNN Deployment through Knowledge Distillation

Chapter 5 addressed the computational efficiency challenge by developing TINED, a novel approach for distilling GNN knowledge into lightweight MLP models. Our analysis revealed that GNN layers contain two distinct types of operations with different properties: feature transformations (FT) that share mathematical formulations with MLP layers, and graph propagation (GP) operations with unique smoothing characteristics.

The TINED framework introduces two key techniques. Teacher injection directly transplants well-trained FT parameters from GNN layers into corresponding MLP layers, providing an effective initialization for knowledge transfer. Dirichlet energy distillation captures the smoothing properties of GP operations and transfers them to the student MLP through specialized loss functions.

Experimental results showed that TINED achieves significant performance improvements while maintaining computational efficiency. The method demonstrated up to $94\times$ speedup compared to the original GNN teacher while often surpassing the teacher’s accuracy, making GNN-level intelligence accessible for deployment in resource-constrained environments.

6.2 Broader Impact and Significance

The contributions of this thesis extend beyond individual algorithmic improvements to address fundamental barriers to GNN adoption in real-world applications. The work provides practical solutions that enable GNNs to be more widely deployed across diverse domains where labeled data is scarce, graph structures are complex, or computational resources are limited.

The stabilized self-training approach makes GNNs viable for applications where obtaining labeled data is expensive or time-consuming, such as in specialized domains like healthcare or scientific research. The heterogeneous graph modeling advances enable better representation of complex real-world systems that involve multiple entity types and relationship patterns. The efficiency improvements through knowledge distillation make it possible to deploy sophisticated graph reasoning capabilities in edge computing environments and real-time applications.

6.3 Limitations and Future Directions

While this thesis makes significant progress in addressing key challenges in GNN applicability, several limitations and opportunities for future work remain.

6.3.1 Current Limitations and Future Directions

Several important directions remain for future research to build upon the contributions presented in this thesis.

For the stabilized self-training approach, there are opportunities to develop more sophisticated pseudo-label selection mechanisms. Future work could explore learnable adaptive confidence thresholds that automatically adjust based on graph characteristics, as well as structure-based pseudo-label selection that considers graph topology. Additionally, applying graph spectral theory to analyze the underlying propagation mechanisms on different graph types would provide deeper theoretical understanding. A comprehensive theoretical analysis of convergence and generalization properties under various graph topologies would strengthen the method’s foundation. The framework could also benefit from integration with graph augmentation, feature augmentation, and label augmentation techniques.

The SlotGAT approach presents several avenues for enhancement. The slot mechanism could be extended to handle dynamic or hierarchical type relationships across different scales, with automatic determination of optimal slot numbers based on graph structure. To improve computational efficiency, future work could focus on selectively choosing important node types for inclusion in the slot mechanism, thereby reducing computational cost. Additionally, developing lightweight slot architectures through parameter sharing and compression techniques would make the approach more practical for large-scale applications.

Expanding TINED to a multi-teacher framework offers a promising avenue for capturing multi-scale structural information and diverse inductive biases, for instance by combining local-aggregation GNNs with long-range Transformers. To resolve potentially conflicting signals among teachers, adaptive integration strategies such as confidence-aware weighting or attention-based fusion could be employed, or a two-stage process could be adopted where teachers first distill to intermediate experts before consolidating their knowledge into a final student MLP.

The current fixed one-to-one layer correspondence between GNN and MLP layers could be made more flexible to better accommodate different architectural designs. A particularly interesting theoretical question is why the properties of Dirichlet energy ratio appear universal across different GNN teachers and datasets. Finally, applying the TINED framework to heterogeneous graphs and other complex graph structures would broaden its applicability.

6.4 Concluding Remarks

This thesis has demonstrated that addressing the fundamental challenges of data scarcity, structural complexity, and computational efficiency is essential for making GNNs more applicable to real-world problems. Through careful analysis of the under-

lying issues and development of targeted solutions, we have shown that it is possible to significantly improve GNN performance and practicality.

The proposed methods—stabilized self-training, slot-based message passing, and teacher injection with Dirichlet energy distillation—represent important steps toward more robust, expressive, and efficient graph neural networks. These contributions provide both theoretical insights and practical tools that can benefit the broader graph learning community and enable new applications of GNNs in diverse domains.

The work presented in this thesis lays the foundation for future research in applicable graph neural networks and demonstrates the importance of addressing practical deployment challenges alongside algorithmic innovations. As graph-structured data becomes increasingly prevalent in various applications, the need for more practical and efficient GNN solutions will continue to grow, making the contributions of this thesis increasingly relevant for the field.

References

- [1] Faraz Ahmed and Muhammad Abulaish. A generic statistical approach for spam detection in online social networks. *Computer Communications*, 36(10-11):1120–1129, 2013.
- [2] Sinan Aral and Paramveer S Dhillon. Social influence maximization under empirical influence models. *Nature human behaviour*, 2(6):375–382, 2018.
- [3] Nurul A Asif, Yeahia Sarker, Ripon K Chakraborty, Michael J Ryan, Md Hafiz Ahamed, Dip K Saha, Faisal R Badal, Sajal K Das, Md Firoz Ali, Sumaya I Moyeen, et al. Graph neural network: A comprehensive review on non-euclidean space. *Ieee Access*, 9:60588–60606, 2021.
- [4] Punam Bedi and Chhavi Sharma. Community detection in social networks. *Wiley interdisciplinary reviews: Data mining and knowledge discovery*, 6(3):115–135, 2016.
- [5] Shai Ben-David, John Blitzer, Koby Crammer, and Fernando Pereira. Analysis of representations for domain adaptation. *Advances in neural information processing systems*, 2006.
- [6] Aleksandar Bojchevski and Stephan Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. In *International Conference on Learning Representations*, 2018.

- [7] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD'08*, pages 1247–1250, 2008.
- [8] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, pages 107–117, 1998.
- [9] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations*, 2014.
- [10] Eliav Buchnik and Edith Cohen. Bootstrapped graph diffusions: Exposing the power of nonlinearity. In *Abstracts of the 2018 ACM International Conference on Measurement and Modeling of Computer Systems*, pages 8–10, 2018.
- [11] Desheng Cai, Shengsheng Qian, Quan Fang, Jun Hu, and Changsheng Xu. User cold-start recommendation via inductive heterogeneous graph neural network. *ACM Transactions on Information Systems*, 2023.
- [12] Yukuo Cen, Xu Zou, Jianwei Zhang, Hongxia Yang, Jingren Zhou, and Jie Tang. Representation learning for attributed multiplex heterogeneous network. In *KDD*, pages 1358–1368, 2019.
- [13] Badrish Chandramouli, Justin J Levandoski, Ahmed Eldawy, and Mohamed F Mokbel. Streamrec: a real-time recommender system. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, 2011.
- [14] Heng-Jui Chang, Shu-wen Yang, and Hung-yi Lee. Distilhubert: Speech representation learning by layer-wise distillation of hidden-unit bert. In *ICASSP*, pages 7087–7091. IEEE, 2022.
- [15] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the

- topological view. In *AAAI Conference on Artificial Intelligence*, pages 3438–3445, 2020.
- [16] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *AAAI*, pages 3438–3445. AAAI Press, 2020.
- [17] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *AAAI*, pages 3438–3445, 2020.
- [18] Lei Chen, Zhengdao Chen, and Joan Bruna. On graph neural networks versus graph-augmented mlps. In *ICLR*, 2021.
- [19] Mengru Chen, Chao Huang, Lianghao Xia, Wei Wei, Yong Xu, and Ronghua Luo. Heterogeneous graph contrastive learning for recommendation. In *Proceedings of the sixteenth ACM international conference on web search and data mining*, 2023.
- [20] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*, pages 1725–1735, 2020.
- [21] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *ICML*, 2020.
- [22] Minmin Chen, Kilian Q. Weinberger, and John Blitzer. Co-training for domain adaptation. In *Conference on Neural Information Processing Systems*, pages 2456–2464, 2011.
- [23] Wei Chen, Yajun Wang, and Siyu Yang. Efficient influence maximization in social networks. In *KDD*, pages 199–208, 2009.

- [24] Dawei Cheng, Yao Zou, Sheng Xiang, and Changjun Jiang. Graph neural networks for financial fraud detection: a review. In *Frontiers of Computer Science*, 2025.
- [25] Fan RK Chung. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997.
- [26] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Conference on Neural Information Processing Systems*, pages 3837–3845, 2016.
- [27] Xiang Deng and Zhongfei Zhang. Graph-free knowledge distillation for graph neural networks. *IJCAI*, 2021.
- [28] Zhihao Ding, Jieming Shi, Qing Li, and Jiannong Cao. Effective illicit account detection on large cryptocurrency multigraphs. In *CIKM*, page 457–466, 2024.
- [29] Xiangyu Dong, Xingyi Zhang, Lei Chen, Mingxuan Yuan, and Sibow Wang. Spacegnn: Multi-space graph neural network for node anomaly detection with extremely limited labels. In *ICLR*, 2025.
- [30] Xiangyu Dong, Xingyi Zhang, Yanni Sun, Lei Chen, Mingxuan Yuan, and Sibow Wang. Smoothggn: Smoothing-aware gnn for unsupervised node anomaly detection. In *Proceedings of the ACM on Web Conference 2025*, pages 1225–1236, 2025.
- [31] Xiangyu Dong, Xingyi Zhang, and Sibow Wang. Rayleigh quotient graph neural networks for graph-level anomaly detection. In *ICLR*, 2024.
- [32] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. metapath2vec: Scalable representation learning for heterogeneous networks. In *KDD '17*, pages 135–144. ACM, 2017.

-
- [33] Ran El-Yaniv and Dmitry Pechyony. Transductive rademacher complexity and its applications. In *Annual Conference on Learning Theory*, pages 157–171, 2007.
- [34] Shaohua Fan, Junxiong Zhu, Xiaotian Han, Chuan Shi, Linmei Hu, Biyu Ma, and Yongliang Li. Metapath-guided heterogeneous graph neural network for intent recommendation. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery and data mining*, 2019.
- [35] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Yihong Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *International World Wide Web Conference*, pages 417–426, 2019.
- [36] Kaituo Feng, Changsheng Li, Ye Yuan, and Guoren Wang. Freekd: Free-direction knowledge distillation for graph neural networks. In *KDD*, 2022.
- [37] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.
- [38] Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3-5):75–174, 2010.
- [39] Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. Protein interface prediction using graph convolutional networks. In *NeurIPS’17*, pages 6530–6539, 2017.
- [40] Sichao Fu, Weifeng Liu, Kai Zhang, Yicong Zhou, and Dapeng Tao. Semi-supervised classification by graph p-laplacian convolutional networks. *Information Sciences*, 560:92–106, 2021.
- [41] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. Magnn: metapath aggregated graph neural network for heterogeneous graph embedding. In *WWW’20*, pages 2331–2341, 2020.

- [42] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *ICML'17*, pages 1263–1272. PMLR, 2017.
- [43] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.
- [44] Yaowen Gu, Si Zheng, Qijin Yin, Rui Jiang, and Jiao Li. Redda: Integrating multiple biological relations to heterogeneous graph neural network for drug-disease association prediction. *Computers in biology and medicine*, 2022.
- [45] Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *AAAI Conference on Artificial Intelligence*, pages 922–929, 2019.
- [46] Arushi Gupta and Rishabh Kaushal. Improving spam detection in online social networks. In *International conference on cognitive computing and information processing*, pages 1–6. IEEE, 2015.
- [47] William L Hamilton. *Graph representation learning*. Morgan & Claypool Publishers, 2020.
- [48] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, 2017.
- [49] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Conference on Neural Information Processing Systems*, pages 1024–1034, 2017.
- [50] Xiaotian Han, Tong Zhao, Yozen Liu, Xia Hu, and Neil Shah. Mlpinit: Embarrassingly simple GNN training acceleration with MLP initialization. In *ICLR*, 2023.

-
- [51] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.
- [52] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:2211.02989*, 2015.
- [53] Huiting Hong, Hantao Guo, Yucheng Lin, Xiaoqing Yang, Zang Li, and Jieping Ye. An attention-based graph neural network for heterogeneous structural learning. In *AAAI’20*, 2020.
- [54] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *Conference on Neural Information Processing Systems*, 2020.
- [55] Yang Hu, Haoxuan You, Zhecan Wang, Zhicheng Wang, Erjin Zhou, and Yue Gao. Graph-mlp: node classification without message passing in graph. *arXiv preprint arXiv:2106.04051*, 2021.
- [56] Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. Gpt-gnn: Generative pre-training of graph neural networks. In *KDD’20*, pages 1857–1867, 2020.
- [57] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer. In *WWW’20*, pages 2704–2710, 2020.
- [58] Yutaro Iiyama, Gianluca Cerminara, Abhijay Gupta, Jan Kieseler, Vladimir Loncar, Maurizio Pierini, Shah Rukh Qasim, Marcel Rieger, Sioni Summers, Gerrit Van Onsem, et al. Distance-weighted graph neural networks on fpgas for real-time particle reconstruction in high energy physics. *Frontiers in big Data*, 2021.

- [59] Aryan Jadon, Avinash Patil, and Shruti Jadon. A comprehensive survey of regression based loss functions for time series forecasting. *arXiv preprint arXiv:2211.02989*, 2022.
- [60] Zhihao Jia, Sina Lin, Rex Ying, Jiaxuan You, Jure Leskovec, and Alex Aiken. Redundancy-free computation for graph neural networks. In *KDD*, 2020.
- [61] Dejun Jiang, Zhenxing Wu, Chang-Yu Hsieh, Guangyong Chen, Ben Liao, Zhe Wang, Chao Shen, Dongsheng Cao, Jian Wu, and Tingjun 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*, 13:1–23, 2021.
- [62] Jing Jiang and ChengXiang Zhai. Instance weighting for domain adaptation in nlp. *ACL*, 2007.
- [63] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling BERT for natural language understanding. In *EMNLP*, pages 4163–4174. Association for Computational Linguistics, 2020.
- [64] Chaitanya K Joshi, Cristian Bodnar, Simon V Mathis, Taco Cohen, and Pietro Lio. On the expressive power of geometric graph neural networks. In *International conference on machine learning*, pages 15330–15355. PMLR, 2023.
- [65] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.
- [66] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.
- [67] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

-
- [68] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *International Conference on Learning Representations*, 2019.
- [69] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *ICLR*, 2019.
- [70] Ryosuke Kojima, Shoichi Ishida, Masateru Ohta, Hiroaki Iwata, Teruki Honma, and Yasushi Okuno. kgcn: a graph-based deep learning framework for chemical structures. *Journal of Cheminformatics*, 12:1–10, 2020.
- [71] Andrea Lancichinetti and Santo Fortunato. Community detection algorithms: a comparative analysis. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics*, 80(5):056117, 2009.
- [72] Dong-Hyun Lee. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on Challenges in Representation Learning, International Conference on Machine Learning*, 2013.
- [73] Minjae Lee and Seoung Bum Kim. Hapgnn: Hop-wise attentive pagerank-based graph neural network. *Information Sciences*, 613:435–452, 2022.
- [74] Seunghyun Lee and Byung Cheol Song. Graph-based knowledge distillation by multi-head attention network. *arXiv preprint arXiv:1907.02226*, 2019.
- [75] Chang Li and Dan Goldwasser. Encoding social information with graph convolutional networks for political perspective detection in news media. In *Association for Computational Linguistics*, pages 2594–2604, 2019.
- [76] Hao-Ting Li, Shih-Chieh Lin, Cheng-Yeh Chen, and Chen-Kuo Chiang. Layer-level knowledge distillation for deep neural network learning. *Applied Sciences*, 9(10):1966, 2019.

- [77] Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. Distance encoding: Design provably more powerful neural networks for graph representation learning. In *NeurIPS*, 2020.
- [78] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI Conference on Artificial Intelligence*, pages 3538–3545, 2018.
- [79] Yayong Li, Jie Yin, and Ling Chen. Informative pseudo-labeling for graph neural networks with few labels. *Data Min Knowl Disc*, 2022.
- [80] Yuchen Li, Ju Fan, Yanhao Wang, and Kian-Lee Tan. Influence maximization on social graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 30(10):1852–1872, 2018.
- [81] Chen Liang, Simiao Zuo, Qingru Zhang, Pengcheng He, Weizhu Chen, and Tuo Zhao. Less is more: Task-aware layer-wise distillation for language model compression. In *ICML*, pages 20852–20867. PMLR, 2023.
- [82] Jian Liang, Jinjun Tang, Fan Gao, Zhe Wang, and Helai Huang. On region-level travel demand forecasting using multi-task adaptive graph attention network. *Information Sciences*, 622:161–177, 2023.
- [83] Jie Liao, Wei Zhou, Fengji Luo, Junhao Wen, Min Gao, Xiuhua Li, and Jun Zeng. Sociallgn: Light graph convolution network for social recommendation. *Information Sciences*, 589:595–607, 2022.
- [84] Zhiqi Lin, Cheng Li, Youshan Miao, Yunxin Liu, and Yinlong Xu. Paragraph: Scaling gnn training on large graphs via computation-aware caching. In *Proceedings of the 11th ACM Symposium on Cloud Computing*, pages 401–415, 2020.

-
- [85] Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. In *ACM SIGKDD Conference on Knowledge Discovery Data Mining*, pages 338–348, 2020.
- [86] Yuankai Luo, Lei Shi, and Xiao-Ming Wu. Classic gnns are strong baselines: Reassessing gnns for node classification. *Advances in Neural Information Processing Systems*, 37:97650–97669, 2024.
- [87] Kha-Dinh Luong and Ambuj Singh. Application of transformers in cheminformatics. *Journal of Chemical Information and Modeling*, 64(11):4392–4409, 2024.
- [88] Qingsong Lv, Ming Ding, Qiang Liu, Yuxiang Chen, Wenzheng Feng, Siming He, Chang Zhou, Jianguo Jiang, Yuxiao Dong, and Jie Tang. Are we really making much progress?: Revisiting, benchmarking and refining heterogeneous graph neural networks. In *KDD '21*, pages 1150–1160, 2021.
- [89] Hehuan Ma, Yu Rong, and Junzhou Huang. Graph neural networks: Scalability. *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 99–119, 2022.
- [90] Rui Miao, Yintao Yang, Yao Ma, Xin Juan, Haotian Xue, Jiliang Tang, Ying Wang, and Xin Wang. Negative samples selecting strategy for graph contrastive learning. *Information Sciences*, 613:667–681, 2022.
- [91] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 5425–5434, 2017.
- [92] Yurii E. Nesterov and Vladimir G. Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, pages 527–566, 2017.

- [93] Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint arXiv:1905.10947*, 2019.
- [94] A James O’malley and Peter V Marsden. The analysis of social networks. *Health services and outcomes research methodology*, 8:222–269, 2008.
- [95] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Conference on Neural Information Processing Systems*, pages 8024–8035, 2019.
- [96] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: online learning of social representations. In *ACM SIGKDD Conference on Knowledge Discovery Data Mining*, pages 701–710, 2014.
- [97] Oleg Platonov, Denis Kuznedelev, Michael Diskin, Artem Babenko, and Liudmila Prokhorenkova. A critical look at the evaluation of gnns under heterophily: Are we really making progress? *ICLR*, 2023.
- [98] Xun Qian, Peter Richtárik, Robert M. Gower, Alibek Sailanbayev, Nicolas Loizou, and Egor Shulgin. SGD with arbitrary sampling: General analysis and improved rates. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 5200–5209, 2019.
- [99] Jiezhong Qiu, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. Deepinf: Social influence prediction with deep learning. In *ACM SIGKDD Conference on Knowledge Discovery Data Mining*, pages 2110–2119, 2018.
- [100] Zhiqiang Que, Hongxiang Fan, Marcus Loo, He Li, Michaela Blott, Maurizio Pierini, Alexander Tapper, and Wayne Luk. Ll-gnn: Low latency graph neural

- networks on fpgas for high energy physics. *ACM Transactions on Embedded Computing Systems*, 2024.
- [101] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding. *Journal of Complex Networks*, 9(2):cnab014, 2021.
- [102] T Konstantin Rusch, Michael M Bronstein, and Siddhartha Mishra. A survey on oversmoothing in graph neural networks. *arXiv preprint arXiv:2303.10993*, 2023.
- [103] Ryoma Sato. A survey on the expressive power of graph neural networks. *arXiv preprint arXiv:2003.04078*, 2020.
- [104] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *ESWC'18*, pages 593–607. Springer, 2018.
- [105] H. J. Scudder. Probability of error of some adaptive pattern-recognition machines. *IEEE Transactions on Information Theory*, pages 363–371, 1965.
- [106] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, pages 93–106, 2008.
- [107] Marco Serafini and Hui Guan. Scalable graph neural network training: The case for sampling. *ACM SIGOPS Operating Systems Review*, 55(1):68–76, 2021.
- [108] Nitai B Silva, Ren Tsang, George DC Cavalcanti, and Jyh Tsang. A graph-based friend recommendation system using genetic algorithm. In *IEEE congress on evolutionary computation*, pages 1–7. IEEE, 2010.
- [109] Tom AB Snijders. Statistical models for social networks. *Annual review of sociology*, 37(1):131–153, 2011.

- [110] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Detecting spammers on social networks. In *Proceedings of the 26th annual computer security applications conference*, pages 1–9, 2010.
- [111] Student. The probable error of a mean. *Biometrika*, pages 1–25, 1908.
- [112] Ke Sun, Zhouchen Lin, and Zhanxing Zhu. Multi-stage self-supervised learning for graph convolutional networks on graphs with few labeled nodes. In *AAAI Conference on Artificial Intelligence*, pages 5892–5899, 2020.
- [113] Enhua Tan, Lei Guo, Songqing Chen, Xiaodong Zhang, and Yihong Zhao. Unik: Unsupervised social network spam detection. In *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*, pages 479–488, 2013.
- [114] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. LINE: large-scale information network embedding. In *International World Wide Web Conference*, pages 1067–1077, 2015.
- [115] Yijun Tian, Shichao Pei, Xiangliang Zhang, Chuxu Zhang, and Nitesh V Chawla. Knowledge distillation on graphs: A survey. *arXiv preprint arXiv:2302.00219*, 2023.
- [116] Yijun Tian, Chuxu Zhang, Zhichun Guo, Xiangliang Zhang, and Nitesh Chawla. Learning mlps on graphs: A unified view of effectiveness, robustness, and efficiency. In *ICLR*, 2022.
- [117] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 2008.
- [118] Fernando Vega-Redondo. *Complex social networks*. Cambridge University Press, 2007.

-
- [119] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.
 - [120] Petar Velickovic, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. Deep graph infomax. In *International Conference on Learning Representations*, 2019.
 - [121] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.
 - [122] Ulrike von Luxburg. A tutorial on spectral clustering. *Stat. Comput.*, 17(4):395–416, 2007.
 - [123] Xinchun Wan, Kaiqiang Xu, Xudong Liao, Yilun Jin, Kai Chen, and Xin Jin. Scalable and efficient full-graph gnn training for large graphs. *Proceedings of the ACM on Management of Data*, 1(2):1–23, 2023.
 - [124] Haorui Wang, Haoteng Yin, Muhan Zhang, and Pan Li. Equivariant and stable positional encoding for more powerful graph neural networks. In *ICLR*, 2022.
 - [125] Hongwei Wang, Fuzheng Zhang, Mengdi Zhang, Jure Leskovec, Miao Zhao, Wenjie Li, and Zhongyuan Wang. Knowledge-aware graph neural networks with label smoothness regularization for recommender systems. In *KDD*, pages 968–977, 2019.
 - [126] Hongwei Wang, Miao Zhao, Xing Xie, Wenjie Li, and Minyi Guo. Knowledge graph convolutional networks for recommender systems. In *WWW’19*, pages 3307–3313, 2019.
 - [127] Mei Wang and Weihong Deng. Deep visual domain adaptation: A survey. *Neurocomputing*, 312:135–153, 2018.

- [128] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.
- [129] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. Kgat: Knowledge graph attention network for recommendation. In *KDD*, pages 950–958, 2019.
- [130] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *WWW’19*, 2019.
- [131] Yifan Wang, Suyao Tang, Yuntong Lei, Weiping Song, Sheng Wang, and Ming Zhang. Disenhan: Disentangled heterogeneous graph attention network for recommendation. In *CIKM ’20*, pages 1605–1614. ACM, 2020.
- [132] Yuyang Wang, Zijie Li, and Amir Barati Farimani. Graph neural networks for molecules. In *Machine learning in molecular sciences*. 2023.
- [133] Zhibo Wang, Jilong Liao, Qing Cao, Hairong Qi, and Zhi Wang. Friendbook: a semantic-based friend recommendation system for social networks. *IEEE transactions on mobile computing*, 14(3):538–551, 2014.
- [134] Oliver Wieder, Stefan Kohlbacher, Melaine Kuenemann, Arthur Garon, Pierre Ducrot, Thomas Seidel, and Thierry Langer. A compact review of molecular property prediction with graph neural networks. *Drug Discovery Today: Technologies*, 2020.
- [135] Lirong Wu, Haitao Lin, Yufei Huang, Tianyu Fan, and Stan Z. Li. Extracting low-/high- frequency knowledge from graph neural networks and injecting it into mlps: An effective gnn-to-mlp distillation framework. In *AAAI*, pages 10351–10360, 2023.

-
- [136] Lirong Wu, Haitao Lin, Yufei Huang, and Stan Z. Li. Quantifying the knowledge in gnn for reliable distillation into mlps. In *ICML*, pages 37571–37581, 2023.
- [137] Xiao-Ming Wu, Zhenguo Li, Anthony Man-Cho So, John Wright, and Shih-Fu Chang. Learning with partially absorbing random walks. In *Conference on Neural Information Processing Systems*, pages 3086–3094, 2012.
- [138] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 2020.
- [139] Bingbing Xu, Junjie Huang, Liang Hou, Huawei Shen, Jinhua Gao, and Xueqi Cheng. Label-consistency based graph neural networks for semi-supervised node classification. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020.
- [140] Xiaoyu Xu, Guansong Pang, Di Wu, and Mingsheng Shang. Joint hyperbolic and euclidean geometry contrastive graph neural networks. *Information Sciences*, 609:799–815, 2022.
- [141] Bencheng Yan, Chaokun Wang, Gaoyang Guo, and Yunkai Lou. Tinygnn: Learning efficient graph neural networks. In *KDD*, 2020.
- [142] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In Yoshua Bengio and Yann LeCun, editors, *ICLR’15*, 2015.
- [143] Carl Yang, Aditya Pal, Andrew Zhai, Nikil Pancha, Jiawei Han, Charles Rosenberg, and Jure Leskovec. Multisage: Empowering gcn with contextualized multi-embeddings on web-scale multipartite networks. In *KDD*, pages 2434–2443, 2020.

- [144] Carl Yang, Yuxin Xiao, Yu Zhang, Yizhou Sun, and Jiawei Han. Heterogeneous network representation learning: A unified framework with survey and benchmark. *IEEE Trans. Knowl. Data Eng.*, 34(10):4854–4873, 2022.
- [145] Cheng Yang, Jiawei Liu, and Chuan Shi. Extract the knowledge of graph neural networks and go beyond it: An effective knowledge distillation framework. In *WWW*, 2021.
- [146] Gang Yang, Xiaofeng Zhang, and Yueping Li. Session-based recommendation with graph neural networks for repeat consumption. In *International Conference on Computing and Pattern*, pages 519–524, 2020.
- [147] Ling Yang, Ye Tian, Minkai Xu, Zhongyi Liu, Shenda Hong, Wei Qu, Wentao Zhang, Bin CUI, Muhan Zhang, and Jure Leskovec. Vqgraph: Rethinking graph representation space for bridging gnns and mlps. In *ICLR*, 2024.
- [148] Yiding Yang, Jiayan Qiu, Mingli Song, Dacheng Tao, and Xinchao Wang. Distilling knowledge from graph convolutional networks. In *CVPR*, 2020.
- [149] Zhen Yang, Ming Ding, Chang Zhou, Hongxia Yang, Jingren Zhou, and Jie Tang. Understanding negative sampling in graph representation learning. In *ACM SIGKDD Conference on Knowledge Discovery Data Mining*, pages 1666–1676, 2020.
- [150] Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. On early stopping in gradient descent learning. *Constructive Approximation*, 26(2):289–315, 2007.
- [151] Jiaxuan You, Rex Ying, and Jure Leskovec. Position-aware graph neural networks. In *ICML*, 2019.
- [152] Liyi Yu, Wangren Qiu, Weizhong Lin, Xiang Cheng, Xuan Xiao, and Jiexia Dai. Hgdti: predicting drug–target interaction by using information aggregation based on heterogeneous graph neural network. *BMC bioinformatics*, 2022.

-
- [153] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks. In *NeurIPS'19*, 2019.
- [154] Bingxu Zhang, Changjun Fan, Shixuan Liu, Kuihua Huang, Xiang Zhao, Jincui Huang, and Zhong Liu. The expressive power of graph neural networks: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2024.
- [155] Chunhui Zhang, Chao Huang, Yijun Tian, Qianlong Wen, Zhongyu Ouyang, Youhuan Li, Yanfang Ye, and Chuxu Zhang. Diving into unified data-model sparsity for class-imbalanced graph representation learning. In *GLFrontiers*, 2022.
- [156] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V Chawla. Heterogeneous graph neural network. In *KDD*, pages 793–803, 2019.
- [157] Dalong Zhang, Xin Huang, Ziqi Liu, Zhiyang Hu, Xianzheng Song, Zhibang Ge, Zhiqiang Zhang, Lin Wang, Jun Zhou, Yang Shuang, et al. Agl: A scalable system for industrial-purpose graph machine learning. *arXiv preprint arXiv:2003.02454*, 2020.
- [158] Ping Zhang and Gary Chartrand. *Introduction to graph theory*, volume 2. Tata McGraw-Hill New York, 2006.
- [159] Shichang Zhang, Yozen Liu, Yizhou Sun, and Neil Shah. Graph-less neural networks: Teaching old mlps new tricks via distillation. In *ICLR*, 2022.
- [160] Tianyu Zhao, Cheng Yang, Yibo Li, Quan Gan, Zhenyi Wang, Fengqi Liang, Huan Zhao, Yingxia Shao, Xiao Wang, and Chuan Shi. Space4hgnn: A novel, modularized and reproducible platform to evaluate heterogeneous graph neural network. In *SIGIR '22*, pages 2776–2789, 2022.
- [161] Wenqing Zheng, Edward W Huang, Nikhil Rao, Sumeet Katariya, Zhangyang Wang, and Karthik Subbian. Cold brew: Distilling graph node representations with incomplete or missing neighborhoods. In *ICLR*, 2022.

- [162] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 2020.
- [163] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.
- [164] Yan Zhou, Murat Kantarcioglu, and Bhavani M. Thuraisingham. Self-training with selection-by-rejection. In *IEEE International Conference on Data Mining*, pages 795–803, 2012.
- [165] Ziang Zhou, Zhihao Ding, Jieming Shi, Shiqi Shen, and Qing Li. Tined: Gnns-to-mlps by teacher injection and dirichlet energy distillation. *ICML*, 2025.
- [166] Ziang Zhou, Jieming Shi, Renchi Yang, Yuanhang Zou, and Qing Li. Slotgat: Slot-based message passing for heterogeneous graphs. *ICML*, 2023.
- [167] Ziang Zhou, Jieming Shi, Shengzhong Zhang, Zengfeng Huang, and Qing Li. Effective stabilized self-training on few-labeled graph data. *Information Sciences*, 2023.
- [168] Meiqi Zhu, Xiao Wang, Chuan Shi, Houye Ji, and Peng Cui. Interpreting and unifying graph neural networks with an optimization framework. In *WWW*, page 1215–1226, 2021.
- [169] Shichao Zhu, Chuan Zhou, Shirui Pan, Xingquan Zhu, and Bin Wang. Relation structure-aware heterogeneous graph neural network. In *ICDM’19*, 2019.
- [170] Martin Zinkevich, Markus Weimer, Alexander J. Smola, and Lihong Li. Parallelized stochastic gradient descent. In *Conference on Neural Information Processing Systems*, pages 2595–2603, 2010.