LEARNING ON GRAPHS WITH ROBUSTNESS, EFFICIENCY AND ADAPTABILITY

A Dissertation

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

by

Kaiwen Dong

Nitesh V. Chawla, Director

Graduate Program in Computer Science and Engineering
Notre Dame, Indiana

April 2025

LEARNING ON GRAPHS WITH
ROBUSTNESS, EFFICIENCY AND ADAPTABILITY

Abstract

by

Kaiwen Dong

Graphs are ubiquitous data structures that capture complex relationships among entities in real-world systems, including social networks, biological networks, transportation systems, and e-commerce platforms. With the rapid growth of graph data, numerous challenges and tasks have emerged, significantly impacting everyday life. While Graph Neural Networks (GNNs) have become the dominant approach for addressing these challenges, they face critical limitations in robustness, efficiency, and adaptability that restrict their broader application.

This thesis systematically addresses these limitations to develop more powerful GNN models. To improve robustness, this thesis investigates GNNs from both model and data perspectives. On the model side, it identifies dataset shift as a fundamental issue and proposes a corresponding solution. On the data side, it introduces a data augmentation method to generate predictive and concise graph representations. To enhance efficiency, the thesis presents techniques that improve GNN expressiveness through efficient graph structural estimation. Additionally, it proposes a novel training paradigm that bypasses the conventional gradient descent process, allowing GNNs to fit directly to data. To improve adaptability, the thesis explores methods for GNNs to generalize to new data and tasks. Specifically, it introduces a framework for link prediction that can adapt to arbitrary graphs during inference. Furthermore, it

develops a practical application of GNNs for predictive tasks in relational databases. By addressing these limitations, this thesis advances the robustness, efficiency, and adaptability of GNNs, expanding their applicability in diverse real-world scenarios.

To my family.

CONTENTS

FIGURES

TABLES

ACKNOWLEDGMENTS

First and foremost, I would like to express my greatest gratitude to my advisor, Dr. Nitesh V. Chawla, for his exceptional supervision and continuous support throughout my Ph.D. journey. His expertise in machine learning and data science, his vision for innovative research, and his passion for discovery have been a constant source of motivation. I am especially grateful that he encouraged me to approach machine learning problems not only from the perspective of an engineer in the industry but also from a more fundamental viewpoint as a research scholar. I deeply appreciate the tremendous freedom he granted me to pursue my interests within graph machine learning. I am also sincerely thankful for his understanding and support for my travels and remote work, which allowed my family to stay together during important moments in our lives. Beyond academics, his thoughtful guidance and warm personality have been invaluable in shaping my personal growth and career development. This dissertation would not have been possible without his steadfast mentorship.

I would like to sincerely thank my thesis committee members, Dr. Xiangliang Zhang, Dr. Meng Jiang, and Dr. Fanny Ye, for serving on my committee. Dr. Zhang encouraged me to look beyond established work and consider the future potential of my research. Dr. Jiang provided detailed comments on my thesis writing, which helped improve its clarity and quality. Dr. Ye offered many insightful suggestions that helped refine the direction of my study. I am truly grateful for their time, support, and contributions to the development of this dissertation.

In addition, I extend my deepest gratitude to my internship mentors, Dr. Kama-

lika Das and Dr. Xiang Gao, at Intuit. Dr. Das has influenced my understanding of how to conduct research that contributes both theoretically and has practical merit in real-world applications. She also provided me the opportunity to carry out research that made a direct impact in production. Dr. Gao guided me in exploring my research interests and developing my professional career. The discussions I shared with them have had a profound impact on me, both personally and professionally.

Furthermore, I want to thank my amazing collaborators: Nitesh Chawla, Zhichun Guo, Kamalika Das, Xiang Gao, Haitao Mao, Yijun Tian, Padmaja Jonnalagedda, Ayan Acharya, Yang Yang, Chuxu Zhang, Khiem Le, David Cieslak, Kai Lu, and Xin Xia. This dissertation would not have been possible without their valuable contributions and support. I would also like to express my gratitude to all my wonderful lab mates in the DIAL Lab, both current and former: Yihong Ma, Joe Germino, Doheon Han, Bruce Huang, Peiyu Li, Jennifer Schnur, Grigorii Khvatskii, Anna Sokol, Daheng Wang, Steven Krieg, and Damien Dablain, as well as Mandana Saebi. Their companionship, discussions, and encouragement have made my Ph.D. journey truly enriching.

Finally, I owe my greatest gratitude to my parents for their unconditional love and endless support. Special thanks to Marsha, who has accompanied my soul both in person and from afar, and who has completed me during the days we have shared and the days that are yet to come. I also wish to thank Marwin, who has helped me understand myself better through the experience of knowing you.

CHAPTER 1

INTRODUCTION

Graphs are fundamental data structures that provide a flexible framework for representing relations among real-world data. Graph-structured data naturally arise in various domains, including social network [91], world wide web [116], protein-protein interactions [141], citation networks [169] and recommender system [85], etc. Beyond conventional machine learning problems where the instances are assumed to be independent, the graph data structure provides a means to capture and analyze the relationship between different instances.

Graph Neural Networks (GNNs) [59, 82, 167] have emerged as the dominant approach for solving graph-related tasks. They have been successfully applied to node-level [82], link-level [178], and graph-level [167] problems across various domains [17, 148]. However, despite their initial success, several challenges limit their broader applicability.

First, the robustness of GNNs is a concern. Their performance can be compromised by low-quality graph data or inherent model limitations, making them less reliable and difficult to trust in real-world applications. Second, GNNs require significant computational resources due to the non-Euclidean nature of graph data. Extracting graph structures and training GNNs on individual graphs demand substantial computation, posing efficiency challenges. Lastly, GNNs have limited generalizability. They struggle to transfer knowledge from one graph to another and are typically confined to tasks where data is already well-represented as a graph.

To solve the aforementioned limitations, this thesis develops threefold techniques

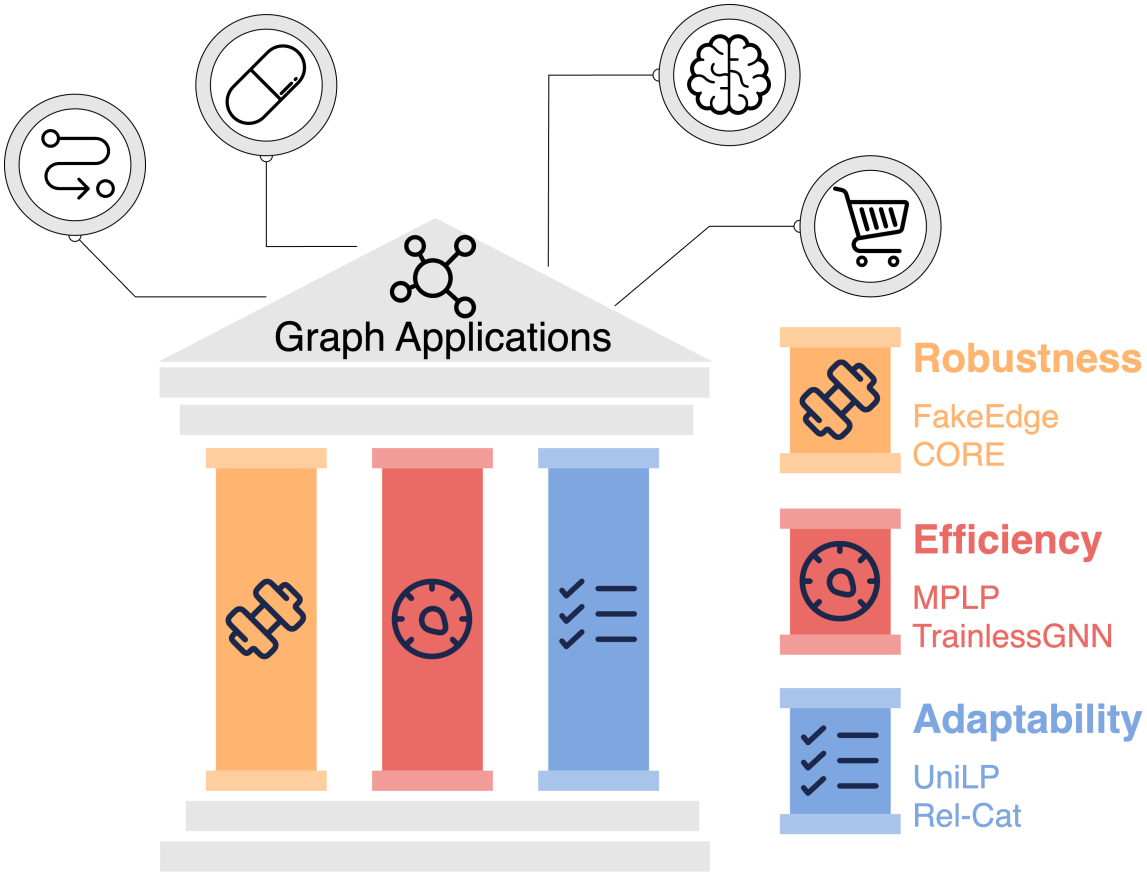

Figure 1.1: Overview of this thesis research. We explore three aspects to enhance GNN's capability on graph applications: robustness, efficiency, and adaptability.

to enhance the GNNs' capability in terms of their robustness, effectiveness, and adaptability. The organization of this thesis is as follows.

In Part I, we improve the robustness of GNNs on link prediction (LP) tasks. Robustness is crucial in GNNs since their performance can significantly decline when faced with low-quality graph data or intrinsic limitations in the model architecture, thereby reducing reliability and trust in practical scenarios. Specifically, two critical aspects necessitate the study of robustness in GNNs for LP. First, GNNs are sensitive to the issue of dataset shift [107], which arises from structural differences between training and testing sets. In LP, the target link influences both the model input

(topological structure) and output (existence or absence of the link), thus causing inconsistent representations between training and testing phases when encoded by GNNs' message passing [57]. Second, the quality and accuracy of collected graph data directly affect LP results. Real-world graphs frequently contain incomplete or incorrect connections due to measurement errors, labeling mistakes, or noisy data collection methods [26, 196]. Such noise makes it difficult for GNNs to distinguish real connections from false ones, severely reducing the reliability and generalization capability of the models. Given these two key challenges from both model and data perspective, it is essential to address and improve the robustness of GNNs to ensure their effective application to real-world problems.

Chapter 2 discusses the robustness of GNNs from a model perspective. This chapter identifies and addresses the dataset shift problem in LP, where the structural patterns of the graph differ significantly between the training and testing phases. The discrepancy arises because target links in the training set are always observed, whereas target links in the testing set are unknown, causing biased node representations encoded by GNNs. This bias results in performance degradation when models are applied to new or unseen graph data. To address this, we proposes FakeEdge, a model-agnostic approach designed to reduce the structural gap between training and testing sets. By deliberately adding or removing the target link, FakeEdge corrects the mismatch and keep the GNNs' link representation consistent across training and testing. The effectiveness and robustness of this method are validated through extensive experiments on multiple datasets spanning diverse backbone models.

Chapter 3 investigates GNNs robustness from a data perspective. This chapter addresses the problem of noisy and incomplete graph data, which often limits the generalization of LP models. It introduces a novel data augmentation framework called COmplete and REduce (CORE). CORE enhances the quality of graph data in two stages: first, the Complete stage adds potentially missing edges to build a

more complete graph, even if it might introduce some noise; second, the Reduce stage removes unnecessary or noisy edges while keeping only the edges essential for effective link prediction. By carefully balancing between adding useful edges and removing harmful ones, CORE improves the quality of the data, resulting in more robust and reliable GNN-based LP models. Extensive experiments demonstrate that CORE achieves superior performance compared to existing methods, validating its effectiveness in producing robust data augmentations for real-world graph tasks.

In Part II, we enhance the efficiency of GNNs to make them more applicable to real-world problems. Due to the non-Euclidean nature of graph data, GNNs typically require considerable computational resources. Extracting structural information and training GNN models on individual graphs involve high computational costs, leading to efficiency challenges. Specifically, two main issues highlight the importance of improving GNN efficiency. First, enhancing the expressive power of GNNs often requires incorporating additional structural information, such as graph substructures [22, 27]. However, identifying and encoding these substructures introduce significant computational overhead, restricting their applicability to large-scale graphs. Second, current GNN approaches typically require training with gradient descent when applied to a graph, significantly increasing computational demands and limiting their practicality. Thus, addressing these efficiency challenges is necessary to ensure GNNs can be effectively utilized in large-scale and resource-sensitive scenarios.

Chapter 4 improves the expressiveness of GNNs while keeping computational complexity manageable. GNNs are provably unable to capture specific graph substructures, which restricts their representational power. For instance, the Common Neighbor substructure [91], critical for LP, cannot be effectively represented by vanilla GNN models [182]. However, explicitly extracting structural features from graphs is computationally expensive, especially at large scales. The core challenge is that LP tasks inherently require link-level structural features, typically resulting in quadratic

computational complexity, whereas vanilla GNNs primarily focus on node-level representations through message-passing. To address this issue, we propose the Message Passing Link Predictor (MPLP), which estimates essential graph substructures without explicitly extracting them. Experimental results demonstrate that MPLP achieves state-of-the-art performance on large-scale LP tasks while maintaining favorable computational efficiency.

Chapter 5 proposes a training-free paradigm for GNNs, significantly enhancing their efficiency in semi-supervised node classification tasks on text-attributed graphs (TAG). Traditional GNNs commonly require iterative optimization methods like gradient descent to update model parameters during training. However, this iterative training approach is computationally expensive. To overcome these limitations, this chapter explores an alternative approach by approximating optimal model parameters directly from node attributes and graph structures, bypassing iterative optimization entirely. Leveraging the observation that node features from the same class cluster in similar linear subspaces, we develop a closed-form solution for fitting linear GNN models. This solution is computationally efficient and aligns with minimum-norm interpolation in an over-parameterized linear regression setting. By removing the need for iterative training, our approach significantly reduces computational costs, making it particularly suitable for practical and scalable applications.

In Part III, we shift our focus to the adaptability of GNNs, investigating whether they can effectively transfer learned knowledge across different graphs and even extend beyond explicitly graph-structured data. Despite their success on individual graph tasks, current GNN approaches typically require full retraining for each new graph, thus ignoring potentially valuable information learned from previously encountered graphs [98]. This limitation becomes particularly significant in scenarios where the new target graph has limited or insufficient training data, making effective learning challenging. Furthermore, GNN research has predominantly concentrated on

problems naturally represented as graphs, such as social or citation networks, while tasks not explicitly structured as graphs have been largely overlooked. Expanding GNN adaptability to these broader scenarios could greatly enhance their practical applicability. Therefore, studying adaptability is important for enabling GNNs to leverage cross-graph knowledge and to extend their application scope beyond traditional graph settings.

Chapter 6 presents an approach to improve the adaptability of GNNs in link prediction tasks, enabling models to transfer knowledge from previously seen graphs and adapt to new graphs during inference without retraining. Typically, GNNs require extensive retraining for each new graph, resulting in limited transferability. To overcome this limitation, this chapter introduces a Universal Link Prediction framework (UniLP) inspired by In-context Learning [18], a strategy widely used in large language models. By employing relevant examples directly within the inference process, the proposed method dynamically adjusts the learned link representations to capture the unique connectivity patterns of each target graph. An attention mechanism is utilized to condition link representations on graph-specific contexts, allowing the model to effectively leverage knowledge from previously seen graphs. Extensive evaluations demonstrate that this approach achieves robust adaptability across diverse graph datasets, matching or even surpassing the performance of traditional methods that require dedicated training for each graph.

Chapter 7 explores the adaptability of GNNs by investigating their applicability to a practical scenario that is not inherently represented as a graph—specifically, a relational database problem. We examine whether GNNs can effectively adapt to the relational data setting without requiring extensive handcrafted feature engineering, which is typically necessary to extract meaningful signals. Transaction-related data commonly resides in relational databases, presenting challenges in directly applying modern machine learning methods due to their complex structure. To ad-

dress this, we propose converting the relational database into a heterogeneous graph structure, enabling the application of GNNs. By doing so, our method leverages the structural relationships inherent within relational databases and enhances the adaptability of GNNs beyond conventional graph datasets. This graph-based approach allows GNNs to automatically utilize relational information without manual feature construction, significantly broadening their practical usability. Experimental results demonstrate that our unified graph formulation successfully adapts GNNs to the relational database setting, achieving superior performance compared to existing production baselines.

In Chapter 8, we conclude this thesis and look into the future of graph machine learning for practical applications.

PART I

ROBUST GNNS

CHAPTER 2

ROBUST GNNS BY ALLEVIATING DATASET SHIFT

2.1 Overview

This chapter investigates the robustness of GNNs from the perspective of addressing dataset shift inherent in their application to link prediction tasks. Despite the widespread success of GNNs, a fundamental issue arises due to their reliance on the assumption that graph connectivity patterns remain consistent between training and testing phases. Specifically, links used in training are always observed, whereas those in testing have not yet formed, creating a gap in structural information and causing biased representations—this phenomenon is termed dataset shift. Through theoretical analysis, we demonstrate that existing GNN-based methods for link prediction are vulnerable to this issue. To mitigate dataset shift, we propose FakeEdge, a model-agnostic technique designed to bridge the topological differences between training and testing graphs. It presents empirical evidence of FakeEdge's effectiveness in addressing the dataset shift problem across diverse datasets and domains.

This chapter primarily builds upon our collaborative study previously published in [37] with Yijun Tian, Zhichun Guo, Yang Yang, and Nitesh V. Chawla.

2.2 Introduction

Graph structured data is ubiquitous across a variety of domains, including social networks [91], protein-protein interactions [141], movie recommendations [85], and citation networks [169]. It provides a non-Euclidean structure to describe the relations

among entities. The link prediction task is to predict missing links or new forming links in an observed network [168]. Recently, with the success of graph neural networks (GNNs) for graph representation learning [19, 33, 43, 82], several GNN-based methods have been developed [81, 90, 118, 157, 178] to solve link prediction tasks. These methods encode the representation of target links with the topological structures and node/edge attributes in their local neighborhood. After recognizing the pattern of observed links (training sets), they predict the likelihood of forming new links between node pairs (testing sets) where no link is yet observed.

Nevertheless, existing methods pose a discrepancy of the target link representation between training and testing sets. As the target link is never observed in the testing set by the nature of the task, it will have a different local topological structure when compared to its counterpart from the training set. Thus, the corrupted topological structure shifts the target link representation in the testing set, which we recognize it as a dataset shift problem [107, 123] in link prediction. Note that there are some existing work [178] applying edge masking to moderate such a problem, similar to our treatment. However, they tend to regard it as an empirical trick and fail to identify the fundamental cause as a problem of dataset shift.

We give a concrete example to illustrate how dataset shift can happen in the link prediction task, especially for GNN-based models with message passing paradigm [57] simulating the 1-dimensional Weisfeiler-Lehman (1-WL) test [160]. In Figure 2.1, we have two local neighborhoods sampled as subgraphs from the training (top) and testing (bottom) set respectively. The node pairs of interest, which we call focal node pairs, are denoted by black bold circles. From a bird's-eye viewpoint, these two subgraphs are isomorphic when we consider the existence of the positive test link (dashed line), even though the test link has not been observed. Ideally, two isomorphic graphs should have the same representation encoded by GNNs, leading to the same link prediction outcome. However, one iteration of 1-WL in Figure 2.1

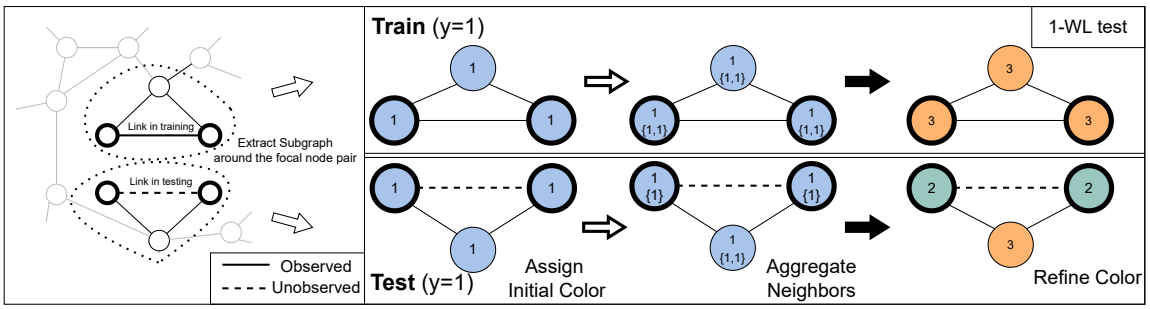

Figure 2.1: 1-WL test is performed to exhibit the learning process of GNNs. Two node pairs (denoted as bold black circles) and their surrounding subgraphs are sampled from the graph as a training (top) and testing (bottom) instance respectively. Two subgraphs are isomorphic when we omit the focal links. One iteration of 1-WL assigns different colors, indicating the occurrence of dataset shift.

produces different colors for the focal node pairs between training and testing sets, which indicates that the one-layer GNN can encode different representations for these two isomorphic subgraphs, giving rise to dataset shift issue.

Dataset shift can substantially degrade model performance since it violates the common assumption that the joint distribution of inputs and outputs stays the same in both the training and testing set. The root cause of this phenomenon in link prediction is the unique characteristic of the target link: the link always plays a dual role in the problem setting and determines both the input and the output for a link prediction task. The existence of the link apparently decides whether it is a positive or negative sample (output). Simultaneously, the presence of the link can also influence how the representation is learned through the introduction of different topological structures around the link (input). Thus, it entangles representation learning and labels in the link prediction problem.

To decouple the dual role of the link, we advocate a framework, namely **subgraph link prediction**, which disentangles the label of the link and its topological structure. As most practical link prediction methods make a prediction by capturing

the local neighborhood of the link [4, 91, 118, 178, 182], we unify them all into this framework, where the input is the extracted subgraph around the focal node pair and the output is the likelihood of forming a link incident with the focal node pair in the subgraph. From the perspective of the framework, we find that the dataset shift issue is mainly caused by the presence/absence of the focal link in the subgraph from the training/testing set. This motivates us to propose a simple but effective technique, **FakeEdge**, to deliberately add or remove the focal link in the subgraph so that the subgraph can stay consistent across training and testing. FakeEdge is a model-agnostic technique, allowing it to be applied to any subgraph link prediction model. It assures that the model would learn the same subgraph representation regardless of the existence of the focal link. Lastly, empirical experiments prove that diminishing the dataset shift issue can significantly boost the link prediction performance on different baseline models.

We summarize our contributions as follows. We first unify most of the link prediction methods into a common framework named as subgraph link prediction, which treats link prediction as a subgraph classification task. In the view of the framework, we theoretically investigate the dataset shift issue in link prediction tasks, which motivates us to propose FakeEdge, a model-agnostic augmentation technique, to ease the distribution gap between the training and testing. We further conduct extensive experiments on a variety of baseline models to reveal the performance improvement with FakeEdge to show its capability of alleviating the dataset shift issue on a broad range of benchmarks.

2.3 Related work

Link Prediction. Early studies on link prediction problems mainly focus on heuristics methods, which require expertise on the underlying trait of network or handcrafted features, including Common Neighbor [91], Adamic–Adar index [4] and Pref-

erential Attachment [9], etc. WLNM [177] suggests a method to encode the induced subgraph of the target link as an adjacency matrix to represent the link. With the huge success of GNN [82], GNN-based link prediction methods have become dominant across different areas. Graph Auto Encoder(GAE) and Variational Graph Auto Encoder(VGAE) [81] perform link prediction tasks by reconstructing the graph structure. SEAL [178] and DE [90] propose methods to label the nodes according to the distance to the focal node pair. To better exploit the structural motifs [105] in distinct graphs, a walk-based pooling method (WalkPool) [118] is designed to extract the representation of the local neighborhood. PLNLP [157] sheds light on pairwise learning to rank the node pairs of interest. Based on two-dimensional Weisfeiler-Lehman tests, Hu et al. propose a link prediction method that can directly obtain node pair representation [63]. To accelerate the inference speed, LLP [58] is proposed to perform the link prediction task by distilling the knowledge from GNNs to MLPs.

Dataset Shift. Dataset shift is a fundamental issue in the world of machine learning. Within the collection of dataset shift issues, there are several specific problems based on which part of the data experience the distributional shift, including covariate shift, concept shift, and prior probability shift. [107] gives a rigorous definition about different dataset shift situations. In the context of GNNs, [170] investigates the generalization ability of GNN models, and propose a self-supervised task to improve the size generalization. [193] studies the problem that the node labels in training set are not uniformly sampled and suggests applying a regularizer to reduce the distributional gap between training and testing. [162] proposes a risk minimization method by exploring multiple context of the observed graph to enable GNNs to generalize to out-of-distribution data. [192] demonstrates that the existing link prediction models can fail to generalize to testing set with larger graphs and designs a structural pairwise embedding to achieve size stability. [13, 20, 101] study the dataset shift

problem for graph-level tasks, especially focusing on the graphs in the training and testing set with varying sizes.

Graph Data Augmentation. Several data augmentation methods are introduced to modify the graph connectivity by adding or removing edges [186]. DropEdge [127] acts like a message passing reducer to tackle over-smoothing or overfitting problems [25]. Topping et al. modify the graph's topological structure by removing negatively curved edges to solve the bottleneck issue [7] of message passing [144]. GDC [84] applies graph diffusion methods on the observed graph to generate a diffused counterpart as the computation graph. For the link prediction task, CFLP [185] generates counterfactual links to augment the original graph. Edge Proposal Set [134] injects edges into the training graph, which are recognized by other link predictors in order to improve performance.

2.4 A proposed unified framework for link prediction

In this section, we formally introduce the link prediction task and formulate several existing GNN-based methods into a common general framework.

2.4.1 Preliminary

Let $\mathcal{G} = (V, E, \boldsymbol{x}^V, \boldsymbol{x}^E)$ be an undirected graph. V is the set of nodes with size n, which can be indexed as $\{i\}_{i=1}^n$. $E \subseteq V \times V$ is the observed set of edges. $\boldsymbol{x}_i^V \in \mathcal{X}^V$ represents the feature of node i. $\boldsymbol{x}_{i,j}^E \in \mathcal{X}^E$ represents the feature of the edge (i, j) if $(i, j) \in E$. The other unobserved set of edges is $E_c \subseteq V \times V \backslash E$, which are either missing or going to form in the future in the original graph $\mathcal{G}$. $d(i, j)$ denotes the shortest path distance between node i and j. The r-hop *enclosing subgraph* $\mathcal{G}_{i,j}^r$ for node i, j is the subgraph induced from $\mathcal{G}$ by node sets $V_{i,j}^r = \{v | v \in V, d(v, i) \leq r \text{ or } d(v, j) \leq r\}$. The edges set of $\mathcal{G}_{i,j}^r$ are $E_{i,j}^r = \{(p, q) | (p, q) \in E \text{ and } p, q \in V_{i,j}^r\}$.

An enclosing subgraph $\mathcal{G}^r_{i,j} = (V^r_{i,j}, E^r_{i,j}, \boldsymbol{x}^V_{V^r_{i,j}}, \boldsymbol{x}^E_{E^r_{i,j}})$ contains all the information in the neighborhood of node i, j. The node set $\{i, j\}$ is called the *focal node pair*, where we are interested in if there exists (observed) or should exist (unobserved) an edge between nodes i, j. In the context of link prediction, we will use the term **subgraph** to denote *enclosing subgraph* in the following sections.

2.4.2 Subgraph link prediction

In this section, we discuss the definition of **Subgraph Link Prediction** and investigate how current link prediction methods can be unified in this framework. We mainly focus on link prediction methods based on GNNs, which propagate the message to each node's neighbors in order to learn the representation. We start by giving the definition of the subgraph's properties:

Definition 1. *Given a graph* $\mathcal{G} = (V, E, \boldsymbol{x}^V, \boldsymbol{x}^E)$ *and the unobserved edge set* E_c*, a subgraph* $\mathcal{G}^r_{i,j}$ *have the following properties:*
1. a ***label*** $\mathrm{y} \in \{0, 1\}$ *of the subgraph indicates if there exists, or will form, an edge incident with focal node pair* $\{i, j\}$*. That is,* $\mathcal{G}^r_{i,j}$ *label* $\mathrm{y} = 1$ *if and only if* $(i, j) \in E \cup E_c$*. Otherwise, label* $\mathrm{y} = 0$*.*
2. the ***existence*** $\mathrm{e} \in \{0, 1\}$ *of an edge in the subgraph indicates whether there is an edge observed at the focal node pair* $\{i, j\}$*. If* $(i, j) \in E$*,* $\mathrm{e} = 1$*. Otherwise* $\mathrm{e} = 0$*.*
3. a ***phase*** $\mathrm{c} \in \{train, test\}$ *denotes whether the subgraph belongs to training or testing stage. Especially for a positive subgraph (*$\mathrm{y} = 1$*), if* $(i, j) \in E$*, then* $\mathrm{c} = train$*. If* $(i, j) \in E_c$*, then* $\mathrm{c} = test$*.*

Note that, the *label* $\mathrm{y} = 1$ does not necessarily indicate the observation of the edge at the focal node pair $\{i, j\}$. A subgraph in the testing set may have the label $\mathrm{y} = 1$ but the edge may not be present. The *existence* $\mathrm{e} = 1$ only when the edge is observed at the focal node pair.

Definition 2. *Given a subgraph $\mathcal{G}_{i,j}^r$,* ***Subgraph Link Prediction*** *is a task to learn a feature* $\mathbf{h}$ *of the subgraph* $\mathcal{G}_{i,j}^r$ *and uses it to predict the label* $\mathrm{y} \in \{0, 1\}$ *of the subgraph.*

Generally, subgraph link prediction regards the link prediction task as a subgraph classification task. The pipeline of subgraph link prediction starts with extracting the subgraph $\mathcal{G}_{i,j}^r$ around the focal node pair $\{i, j\}$, and then applies GNNs to encode the node representation $\boldsymbol{Z}$. The latent feature $\mathbf{h}$ of the subgraph is obtained by pooling methods on $\boldsymbol{Z}$. In the end, the subgraph feature $\mathbf{h}$ is fed into a classifier. In summary, the whole pipeline entails:

1. **Subgraph Extraction**: Extract the subgraph $\mathcal{G}_{i,j}^r$ around the focal node pair $\{i, j\}$;
2. **Node Representation Learning**: $\boldsymbol{Z} = \texttt{GNN}(\mathcal{G}_{i,j}^r)$, where $\boldsymbol{Z} \in \mathbb{R}^{|V_{i,j}^r| \times F_{\text{hidden}}}$ is the node embedding matrix learned by the GNN encoder;
3. **Pooling**: $\mathbf{h} = \texttt{Pooling}(\boldsymbol{Z}; \mathcal{G}_{i,j}^r)$, where $\mathbf{h} \in \mathbb{R}^{F_{\text{pooled}}}$ is the latent feature of the subgraph $\mathcal{G}_{i,j}^r$;
4. **Classification**: $\mathrm{y} = \texttt{Classifier}(\mathbf{h})$.

There are two main streams of GNN-based link prediction models. Models like SEAL [178] and WalkPool [118] can naturally fall into the subgraph link prediction framework, as they thoroughly follow the pipeline. In SEAL, SortPooling [181] serves as a readout to aggregate the node's features in the subgraph. WalkPool designs a random-walk based pooling method to extract the subgraph feature $\mathbf{h}$. Both methods take advantage of the node's representation from the entire subgraph.

In addition, there is another stream of link prediction models, such as GAE [81] and PLNLP [157], which learns the node representation and then devises a score function on the representation of the focal node pair to represent the likelihood of forming a link. We find that these GNN-based methods with the message passing paradigm also belong to a subgraph link prediction task. Considering a GAE with

l layers, each node v essentially learns its embedding from its l-hop neighbors $\{i|i \in V, d(i,v) \leq l\}$. The score function can be then regarded as a center pooling on the subgraph, which only aggregates the features of the focal node pair as $\mathbf{h}$ to represent the subgraph. For a focal node pair $\{i,j\}$ and GAE with l layers, an l-hop subgraph $\mathcal{G}^l_{i,j}$ sufficiently contains all the information needed to learn the representation of nodes in the subgraph and score the focal node pair $\{i,j\}$. Thus, the GNN-based models can also be seen as a citizen of subgraph link prediction. In terms of the score function, there are plenty of options depending on the predictive power in practice. In general, the common choices are: (1) Hadamard product: $\mathbf{h} = z_i \circ z_j$; (2) MLP: $\mathbf{h} = \texttt{MLP}(z_i \circ z_j)$ where $\texttt{MLP}$ is the Multi-Layer Perceptron; (3) BiLinear: $\mathbf{h} = z_i \boldsymbol{W} z_j$ where $\boldsymbol{W}$ is a learnable matrix; (4) BiLinearMLP: $\mathbf{h} = \texttt{MLP}(z_i) \circ \texttt{MLP}(z_j)$.

In addition to GNN-based methods, the concept of the subgraph link prediction can be extended to low-order heuristics link predictors, like Common Neighbor [91], Adamic–Adar index [4], Preferential Attachment [9], Jaccard Index [69], and Resource Allocation [191]. The predictors with the order r can be computed by the subgraph $\mathcal{G}^r_{i,j}$. The scalar value can be seen as the latent feature $\mathbf{h}$.

2.5 FakeEdge: Mitigates dataset shift in subgraph link prediction

In this section, we start by giving the definition of dataset shift in the general case, and then formally discuss how dataset shift occurs with regard to subgraph link prediction. Then we propose FakeEdge as a graph augmentation technique to ease the distribution gap of the subgraph representation between the training and testing sets. Lastly, we discuss how FakeEdge can enhance the expressive power of any GNN-based subgraph link prediction model.

2.5.1 Dataset shift

Definition 3. ***Dataset Shift*** *happens when the joint distribution between train and test is different. That is,* $p(\mathbf{h}, \mathrm{y}|\mathrm{c} = train) \neq p(\mathbf{h}, \mathrm{y}|\mathrm{c} = test)$.

A simple example of dataset shift is an object detection system. If the system is only designed and trained under good weather conditions, it may fail to capture objects in bad weather. In general, dataset shift is often caused by some unknown latent variable, like the weather condition in the example above. The unknown variable is not observable during the training phase so the model cannot fully capture the conditions during testing. Similarly, the edge existence $\mathrm{e} \in \{0, 1\}$ in the subgraph poses as an "unknown" variable in the subgraph link prediction task. Most of the current GNN-based models neglect the effect of the edge existence on encoding the subgraph's feature.

Definition 4. *A subgraph's feature* $\mathbf{h}$ *is called Edge Invariant if* $p(\mathbf{h}, \mathrm{y}|\mathrm{e}) = p(\mathbf{h}, \mathrm{y})$.

To explain, the **Edge Invariant** subgraph embedding stays the same no matter if the edge is present at the focal node pair or not. It disentangles the edge's existence and the subgraph representation learning. For example, common neighbor predictor is Edge Invariant because the existence of an edge at the focal node pair will not affect the number of common neighbors that two nodes can have. However, Preferential Attachment, another widely used heuristics link prediction predictor, is **not** Edge Invariant because the node degree varies depending on the existence of the edge.

Theorem 1. `GNN` *cannot learn the subgraph feature* $\mathbf{h}$ *to be Edge Invariant.*

Recall that the subgraphs in Figure 2.1 are encoded differently between the training and testing set because of the presence/absence of the focal link. Thus, the vanilla GNN cannot learn the Edge Invariant subgraph feature. Learning Edge Invariant subgraph feature is crucial to mitigate the dataset shift problem. Here, we give our main theorem about the issue in the link prediction task:

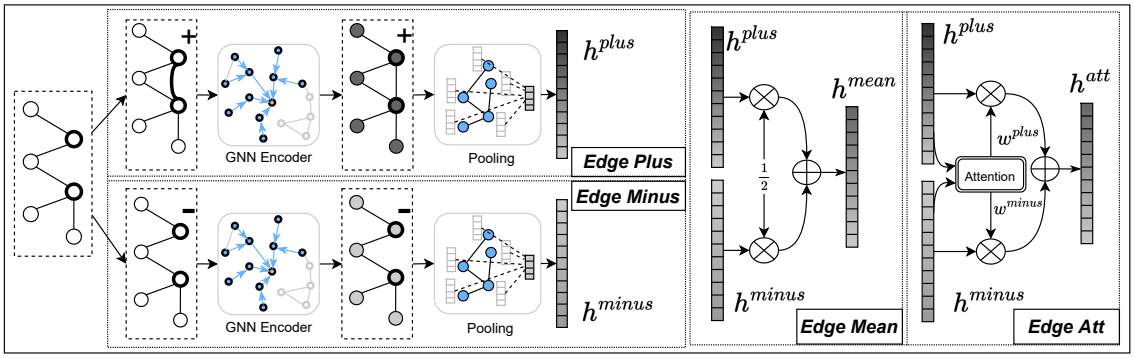

Figure 2.2: The proposed four FakeEdge methods. In general, FakeEdge encourages the link prediction model to learn the subgraph representation by always deliberately adding or removing the edges at the focal node pair in each subgraph. In this way, FakeEdge can reduce the distribution gap of the learned subgraph representation between the training and testing set.

Theorem 2. *Given* $p(\mathbf{h}, \mathrm{y}|\mathrm{e}, \mathrm{c}) = p(\mathbf{h}, \mathrm{y}|\mathrm{e})$*, there is no* ***Dataset Shift*** *in the link prediction if the subgraph embedding is* ***Edge Invariant****. That is,* $p(\mathbf{h}, \mathrm{y}|\mathrm{e}) = p(\mathbf{h}, \mathrm{y}) \Longrightarrow p(\mathbf{h}, \mathrm{y}|\mathrm{c}) = p(\mathbf{h}, \mathrm{y})$.

The assumption $p(\mathbf{h}, \mathrm{y}|\mathrm{e}, \mathrm{c}) = p(\mathbf{h}, \mathrm{y}|\mathrm{e})$ states that when the edge at the focal node pair is taken into consideration, the joint distribution keeps the same across the training and testing stages, which means that there is no other underlying unobserved latent variable shifting the distribution. The theorem shows an **Edge Invariant** subgraph embedding will not cause a dataset shift phenomenon.

Theorem 2 gives us the motivation to design the subgraph embedding to be Edge Invariant. When it comes to GNNs, the practical GNN is essentially a message passing neural network [57]. The existence of the edge incident at the focal node pair can determine the computational graph for message passing when learning the node representation.

2.5.2 Proposed methods

Having developed conditions of dataset shift phenomenon in link prediction, we next introduce a collection of subgraph augmentation techniques named as **FakeEdge** (Figure 2.2), which satisfies the conditions in Theorem 2. The motivation is to mitigate the distribution shift of the subgraph embedding by eliminating the different patterns of target link existence between training and testing sets. Note that all of the strategies follow the same discipline: align the topological structure around the focal node pair in the training and testing datasets, especially for the isomorphic subgraphs. Therefore, we expect that it can gain comparable performance improvement across different strategies.

Compared to the vanilla GNN-based subgraph link prediction methods, FakeEdge augments the computation graph for node representation learning and subgraph pooling step to obtain an Edge Invariant embedding for the entire subgraph.

Edge Plus A simple strategy is to always make the edge present at the focal node pair for all training and testing samples. Namely, we add an edge into the edge set of subgraph by $E^{r+}_{i,j} = E^{r}_{i,j} \cup \{(i,j)\}$, and use this edge set to calculate the representation $\mathbf{h}^{plus}$ of the subgraph $\mathcal{G}^{r+}_{i,j}$.

Edge Minus Another straightforward modification is to remove the edge at the focal node pair if existing. That is, we remove the edge from the edge set of subgraph by $E^{r-}_{i,j} = E^{r}_{i,j} \backslash \{(i,j)\}$, and obtain the representation $\mathbf{h}^{minus}$ from $\mathcal{G}^{r-}_{i,j}$.

For GNN-based models, adding or removing edges at the focal node pair can amplify or reduce message propagation along the subgraph. It may also change the connectivity of the subgraph. We are interested to see if it can be beneficial to take both situations into consideration by combining them. Based on *Edge Plus* and *Edge Minus*, we further develop another two Edge Invariant methods:

Edge Mean To combine *Edge Plus* and *Edge Minus*, one can extract these two features and fuse them into one view. One way is to take the average of the two

latent features by $\mathbf{h}^{mean} = \frac{\mathbf{h}^{plus}+\mathbf{h}^{minus}}{2}$.

Edge Att *Edge Mean* weighs $\mathcal{G}_{i,j}^{r+}$ and $\mathcal{G}_{i,j}^{r-}$ equally on all subgraphs. To vary the importance of two modified subgraphs, we can apply an adaptive weighted sum operation. Similar to the practice in the text translation [95], we apply an attention mechanism to fuse the $\mathbf{h}^{plus}$ and $\mathbf{h}^{minus}$ by:

$$\mathbf{h}^{att} = w^{plus} * \mathbf{h}^{plus} + w^{minus} * \mathbf{h}^{minus}, \tag{2.1}$$

$$\text{where } w^{\cdot} = \text{SoftMax}(\boldsymbol{q}^{\intercal} \cdot tanh(\boldsymbol{W} \cdot \mathbf{h}^{\cdot} + \boldsymbol{b})) \tag{2.2}$$

2.5.3 Expressive power of structural representation

In addition to solving the issue of dataset shift, FakeEdge can tackle another problem that impedes the expressive power of link prediction methods on the structural representation [137]. In general, a powerful model is expected to discriminate most of the non-isomorphic focal node pairs. For instance, in Figure 2.3 we have two isomorphic subgraphs A and B, which do not have any overlapping nodes. Suppose that the focal node pairs we are interested in are $\{u, w\}$ and $\{v, w\}$. Obviously, those two focal node pairs have different structural roles in the graph, and we expect different structural representations for them. With GNN-based methods like GAE, the node representation of the node u and v will be the same $z_u = z_v$, due to the fact that they have isomorphic neighborhoods. GAE applies a score function on the focal node pair to pool the subgraph's feature. Hence, the structural representation of node sets $\{u, w\}$ and $\{v, w\}$ would be the same, leaving them inseparable in the embedding space. This issue is caused by the limitation of GNNs, whose expressive power is bounded by 1-WL test [167].

Zhang et al. address this problem by assigning distinct labels between the focal node pair and the rest of the nodes in the subgraph [182]. FakeEdge manages to resolve the issue by augmenting the neighborhoods of those two isomorphic nodes. For

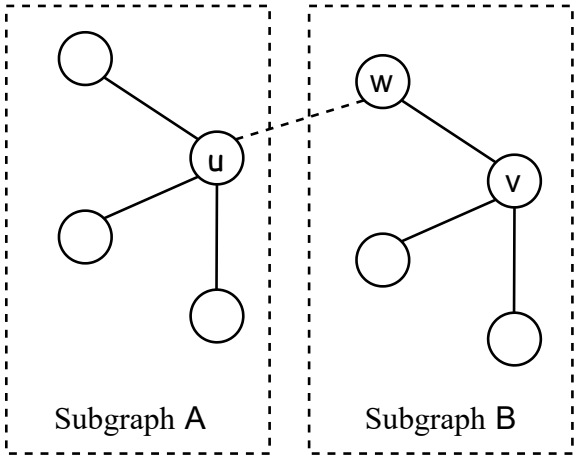

Figure 2.3: Given two isomorphic but non-overlapping subgraphs A and B, GNNs learn the same representation for the nodes u and v. Hence, GNN-based methods cannot distinguish focal node pairs $\{u, w\}$ and $\{v, w\}$. However, by adding a FakeEdge at $\{u, w\}$ (shown as the dashed line in the figure), it can break the tie of the representation for u and v, thanks to u's modified neighborhood.

instance, we can utilize the *Edge Plus* strategy to deliberately add an edge between nodes u and w (shown as the dashed line in Figure 2.3). Note that the edge between v and w has already existed. There is no need to add an edge between them. Therefore, the node u and v will have different neighborhoods (u has 4 neighbors and v has 3 neighbors), resulting in the different node representation between the node u and v after the first iteration of message propagation with GNN. In the end, we can obtain different representations for two focal node pairs. Other FakeEdge methods like *Edge Minus* can also tackle the issue in a similar way.

According to Theorem 2 in [182], such non-isomorphic focal node pairs $\{u, w\}$, $\{v, w\}$ are not sporadic cases in a graph. Given an n nodes graph whose node degree is $\mathcal{O}(\log^{\frac{1-\epsilon}{2r}} n)$ for any constant $\epsilon > 0$, there exists $\omega(n^{2\epsilon})$ pairs of such kind of $\{u, w\}$ and $\{v, w\}$, which cannot be distinguished by GNN-based models like GAE. However, FakeEdge can enhance the expressive power of link prediction methods by modifying the subgraph's local connectivity.

2.6 Experiments

In this section, we conduct extensive experiments to evaluate how FakeEdge can mitigate the dataset shift issue on various baseline models in the link prediction task. Then we empirically show the distribution gap of the subgraph representation between the training and testing and discuss how the dataset shift issue can worsen with deeper GNNs. The code for the experiment can be found at `https://github.com/Barcavin/FakeEdge`.

2.6.1 Experimental setup

Baseline methods. We show how FakeEdge techniques can improve the existing link prediction methods, including GAE-like models [81], PLNLP [157], SEAL [178], and WalkPool [118]. To examine the effectiveness of FakeEdge, we compare the model performance with subgraph representation learned on the **original** unmodified subgraph and the FakeEdge augmented ones. For GAE-like models, we apply different GNN encoders, including GCN [82], SAGE [59] and GIN [167]. SEAL and WalkPool have already been implemented in the fashion of the subgraph link prediction. However, a subgraph extraction preprocessing is needed for GAE and PLNLP, since they are not initially implemented as the subgraph link prediction. GCN, SAGE, and PLNLP use a score function to pool the subgraph. GCN and SAGE use the Hadamard product as the score function, while MLP is applied for PLNLP (see Section 2.4.2 for discussions about the score function). Moreover, GIN applies a subgraph-level pooling strategy, called "mean readout" [167], whose pooling is based on the entire subgraph. Similarly, SEAL and WalkPool also utilize the pooling on the entire subgraph to aggregate the representation. More details about the model implementation can be found in section A.3.

TABLE 2.1

COMPARISON WITH AND WITHOUT FAKEEDGE.

Models	FakeEdge	Cora	Citeseer	Pubmed	USAir	NS	PB	Yeast	C.ele	Power	Router	E.coli
GCN	*Original*	84.92±1.95	77.05±2.18	81.58±4.62	94.07±1.50	96.92±0.73	93.17±0.45	93.76±0.65	88.78±1.85	76.32±4.65	60.72±5.88	95.35±0.36
	Edge Plus	91.94±0.90	89.54±1.17	97.91±0.14	97.10±1.01	98.03±0.72	95.48±0.42	97.86±0.27	89.65±1.74	**85.42±0.91**	95.96±0.41	98.05±0.30
	Edge Minus	92.01±0.94	**90.29±0.88**	97.87±0.15	97.16±0.97	**98.14±0.66**	95.50±0.43	**97.90±0.29**	89.47±1.86	85.39±1.08	96.05±0.37	97.97±0.31
	Edge Mean	91.86±0.76	89.61±0.96	97.94±0.13	97.19±1.00	98.08±0.66	**95.52±0.43**	97.70±0.36	89.62±1.82	85.23±1.00	**96.08±0.35**	**98.07±0.27**
	Edge Att	**92.06±0.85**	88.96±1.05	**97.96±0.12**	**97.20±0.69**	97.96±0.39	95.46±0.45	97.65±0.17	**89.76±2.06**	85.26±1.32	95.90±0.47	98.04±0.16
SAGE	*Original*	89.12±0.90	87.76±0.97	94.95±0.44	96.57±0.57	98.11±0.48	94.12±0.45	97.11±0.31	87.62±1.63	79.35±1.66	88.37±1.46	95.70±0.44
	Edge Plus	93.21±0.82	90.88±0.80	97.91±0.14	97.64±0.73	**98.72±0.59**	95.68±0.39	98.20±0.13	**90.94±1.48**	86.36±0.97	**96.46±0.38**	98.41±0.19
	Edge Minus	92.45±0.78	90.14±1.04	97.93±0.14	97.50±0.67	98.66±0.55	95.57±0.39	98.13±0.10	90.83±1.59	85.62±1.17	92.91±1.09	98.34±0.26
	Edge Mean	92.77±0.69	90.60±0.94	97.96±0.13	**97.67±0.70**	98.62±0.61	**95.69±0.37**	98.20±0.13	90.86±1.51	86.24±1.01	96.22±0.38	98.41±0.21
	Edge Att	**93.31±1.02**	**91.01±1.14**	**98.01±0.13**	97.40±0.94	98.70±0.59	95.49±0.49	**98.22±0.24**	90.64±1.88	**86.46±0.91**	96.31±0.59	**98.43±0.13**
GIN	*Original*	82.70±1.93	77.85±2.64	91.32±1.13	94.89±0.89	96.05±1.10	92.95±0.51	94.50±0.65	85.23±2.56	73.29±3.88	84.29±1.20	94.34±0.57
	Edge Plus	90.72±1.11	89.54±1.19	**97.63±0.14**	96.03±1.37	98.51±0.55	95.38±0.35	**97.84±0.40**	**89.71±2.06**	**86.61±0.87**	**95.79±0.48**	97.67±0.23
	Edge Minus	89.88±1.26	89.30±1.08	97.27±0.17	96.36±0.83	98.62±0.45	95.35±0.35	97.80±0.41	89.40±1.91	86.55±0.83	95.72±0.45	97.33±0.36
	Edge Mean	90.30±1.22	89.47±1.13	97.53±0.19	**96.45±0.90**	**98.66±0.45**	**95.39±0.37**	97.78±0.40	89.66±2.00	86.51±0.92	95.73±0.43	97.57±0.32
	Edge Att	**90.76±0.88**	**89.55±0.61**	97.50±0.15	96.34±0.82	98.35±0.54	95.29±0.29	97.66±0.33	89.39±1.61	86.21±0.67	95.78±0.52	**97.74±0.33**
PLNLP	*Original*	82.37±1.70	82.93±1.73	87.36±4.90	95.37±0.87	97.86±0.93	92.99±0.71	95.09±1.47	88.31±2.21	81.59±4.31	86.41±1.63	90.63±1.68
	Edge Plus	91.62±0.87	**89.88±1.19**	98.31±0.21	98.09±0.73	**98.77±0.39**	**95.33±0.39**	98.10±0.33	**91.77±2.16**	90.04±0.57	**96.45±0.40**	**98.03±0.23**
	Edge Minus	**91.84±1.42**	88.99±1.48	**98.44±0.14**	97.92±0.52	98.59±0.44	95.20±0.34	98.01±0.38	91.60±2.23	89.26±0.58	95.01±0.47	97.80±0.16
	Edge Mean	91.77±1.49	89.45±1.50	98.36±0.16	**98.17±0.60**	98.66±0.56	95.30±0.37	**98.10±0.39**	91.70±2.18	90.05±0.52	96.29±0.47	98.02±0.20
	Edge Att	91.22±1.34	88.75±1.70	98.41±0.17	98.13±0.61	98.70±0.40	95.32±0.38	98.06±0.37	91.72±2.12	**90.08±0.54**	96.40±0.40	98.01±0.18
SEAL	*Original*	90.13±1.94	87.59±1.57	95.79±0.78	97.26±0.58	97.44±1.07	95.06±0.46	96.91±0.45	88.75±1.90	78.14±3.14	92.35±1.21	97.33±0.28
	Edge Plus	90.01±1.95	89.65±1.22	97.30±0.34	97.34±0.59	98.35±0.63	95.35±0.38	97.67±0.32	89.20±1.86	85.25±0.80	95.47±0.58	97.84±0.25
	Edge Minus	91.04±1.91	89.74±1.16	97.50±0.33	97.27±0.63	98.17±0.74	**95.36±0.37**	97.64±0.30	89.35±1.98	**85.30±0.91**	**95.77±0.79**	97.79±0.30
	Edge Mean	90.36±2.17	**89.87±1.14**	**97.52±0.34**	**97.38±0.68**	98.23±0.70	95.30±0.34	97.68±0.33	89.19±1.85	85.30±0.87	95.61±0.64	97.83±0.23
	Edge Att	**91.08±1.67**	89.35±1.43	97.26±0.45	97.04±0.79	**98.52±0.57**	95.19±0.43	**97.70±0.40**	**89.37±1.40**	85.24±1.39	95.14±0.62	**97.90±0.33**
WalkPool	*Original*	**92.00±0.79**	**89.64±1.01**	97.70±0.19	97.83±0.97	99.00±0.45	94.53±0.44	96.81±0.92	93.71±1.11	82.43±3.57	87.46±7.45	95.00±0.90
	Edge Plus	91.96±0.79	89.49±0.96	98.36±0.13	97.97±0.96	98.99±0.58	95.47±0.32	98.28±0.24	93.79±1.11	91.24±0.84	97.31±0.26	98.65±0.17
	Edge Minus	91.97±0.80	89.61±1.04	**98.43±0.10**	98.03±0.95	99.02±0.54	95.47±0.32	**98.30±0.23**	**93.83±1.13**	**91.28±0.90**	**97.35±0.28**	98.66±0.17
	Edge Mean	91.77±0.74	89.55±1.09	98.39±0.11	98.01±0.89	99.02±0.56	95.47±0.29	98.30±0.24	93.70±1.12	91.26±0.81	97.27±0.29	98.65±0.19
	Edge Att	91.98±0.80	89.36±0.74	98.37±0.19	**98.12±0.81**	**99.03±0.50**	**95.47±0.27**	98.28±0.24	93.63±1.11	91.25±0.60	97.27±0.27	**98.70±0.14**

The best results (AUC) are highlighted in bold..

Benchmark datasets. For the experiment, we use 3 datasets with node attributes and 8 without attributes. The graph datasets with node attributes are three citation networks: **Cora** [102], **Citeseer** [56], and **Pubmed** [109]. The graph datasets without node attributes are eight graphs in a variety of domains: **USAir** [10], **NS** [110], **PB** [3], **Yeast** [150], **C.ele** [158], **Power** [158], **Router** [136], and **E.coli** [180]. More details about the benchmark datasets can be found in section A.4.

Evaluation protocols. Following the same experimental setting as of [118, 178], the links are split into 3 parts: 85% for training, 5% for validation, and 10% for testing. The links in validation and testing are unobserved during the training phase. We also implement a universal data pipeline for different methods to eliminate the data perturbation caused by train/test split. We perform 10 random data splits to reduce the performance disturbance. Area under the curve (AUC) [14] is used as the evaluation metrics and is reported by the epoch with the highest score on the validation set.

2.6.2 Results

FakeEdge on GAE-like models. The results of models with (*Edge Plus*, *Edge Minus*, *Edge Mean*, and *Edge Att*) and without (*Original*) FakeEdge are shown in Table 2.1. We observe that FakeEdge is a vital component for all different methods. With FakeEdge, the link prediction model can obtain a significant performance improvement on all datasets. GAE-like models and PLNLP achieve the most remarkable performance improvement when FakeEdge alleviates the dataset shift issue. FakeEdge boosts them by 2%-11% on different datasets. GCN, SAGE, and PLNLP all have a score function as the pooling methods, which is solely based on the focal node pair. In particular, the focal node pair is incident with the target link, which determines how the message passes around it. Therefore, the most severe dataset

shift issues happen at the embedding of the focal node pair during the node representation learning step. FakeEdge is expected to bring a notable improvement to these situations.

Encoder matters. In addition, the choice of encoder plays an important role when GAE is deployed on the *Original* subgraph. We can see that SAGE shows the best performance without FakeEdge among these 3 encoders. However, after applying FakeEdge, all GAE-like methods achieve comparable better results regardless of the choice of the encoder. We come to a hypothesis that the plain SAGE itself leverages the idea of FakeEdge to partially mitigate the dataset shift issue. Each node's neighborhood in SAGE is a fixed-size set of nodes, which is uniformly sampled from the full neighborhood set. Thus, when learning the node representation of the focal node pair in the positive training sets, it is possible that one node of the focal node pair is not selected as the neighbor of the other node during the neighborhood sampling stage. In this case, the FakeEdge technique *Edge Minus* is applied to modify such a subgraph.

FakeEdge on subgraph-based models. In terms of SEAL and WalkPool, FakeEdge can still robustly enhance the model performance across different datasets. Especially for datasets like Power and Router, FakeEdge increases the AUC by over 10% on both methods. Both methods achieve better results across different datasets, except WalkPool model on datasets Cora and Citeseer. One of the crucial components of WalkPool is the walk-based pooling method, which actually operates on both the *Edge Plus* and *Edge Minus* graphs. Different from FakeEdge technique, WalkPool tackles the dataset shift problem mainly on the subgraph pooling stage. Thus, WalkPool shows similar model performance between the *Original* and FakeEdge augmented graphs. Moreover, SEAL and WalkPool have utilized one of the

FakeEdge techniques as a trick in their initial implementations. However, they have failed to explicitly point out the fix of dataset shift issue from such a trick in their papers.

Different FakeEdge techniques. When comparing different FakeEdge techniques, *Edge Att* appears to be the most stable, with a slightly better overall performance and a smaller variance. However, there is no significant difference between these techniques. This observation is consistent with our expectation since all FakeEdge techniques follow the same discipline to fix the dataset shift issue.

2.6.3 Further discussions

In this section, we conduct experiments to more thoroughly study why FakeEdge can improve the performance of the link prediction methods. We first give an empirical experiment to show how severe the distribution gap can be between training and testing. Then, we discuss the dataset shift issue with deeper GNNs. Last but not the least, we explore how FakeEdge can even improve the performance of heuristics predictors.

2.6.4 Distribution gap between the training and testing

FakeEdge aims to produce Edge Invariant subgraph embedding during the training and testing phases in the link prediction task, especially for those positive samples $p(\mathbf{h}|\mathrm{y}=1)$. That is, the subgraph representation of the positive samples between the training and testing should be difficult, if at all, to be distinguishable from each other. Formally, we ask whether $p(\mathbf{h}|\mathrm{y}=1,\mathrm{c}=\mathrm{train}) = p(\mathbf{h}|\mathrm{y}=1,\mathrm{c}=\mathrm{test})$, by conducting an empirical experiment on the subgraph embedding.

We retrieve the subgraph embedding of the positive samples from both the training and testing stages, and randomly shuffle the embedding. Then we classify whether

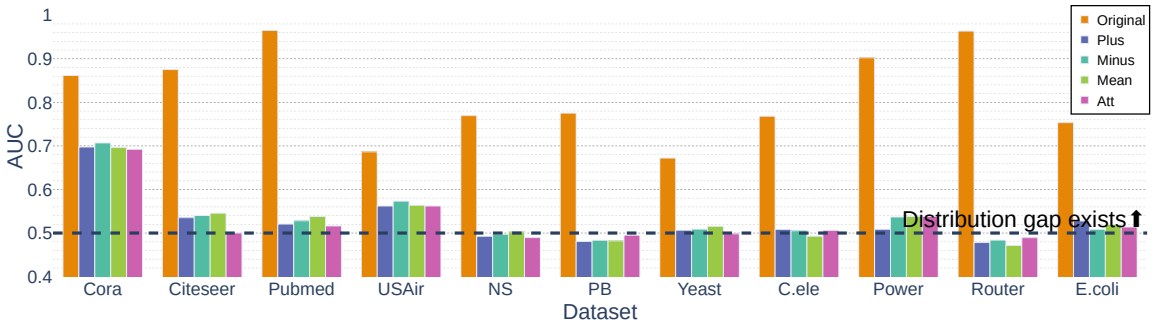

Figure 2.4: Distribution gap (AUC) of the positive samples between the training and testing set.

the sample is from training (c = train) or testing (c = test). The shuffled positive samples are split 80%/20% as train and inference sets. Note that the train set here, as well as the inference set, contains both the shuffled positive samples from the training and testing set in the link prediction task. Then we feed the subgraph embedding into a 2-layer MLP classifier to investigate whether the classifier can differentiate the training samples (c = train) and the testing samples (c = test). In general, the classifier will struggle to undertake the classification if the embedding of training and testing samples is drawn from the same underlying distribution, which indicates there is no significant dataset shift issue.

We use GAE with the GCN as the encoder to run the experiment. AUC is used to measure the discriminating power of the classifier. The results are shown in Figure 2.4. Without FakeEdge, the classifier shows a significant ability to separate positive samples between training and testing. When it comes to the subgraph embedding with FakeEdge, the classifier stumbles in distinguishing the samples. The comparison clearly reveals how different the subgraph embedding can be between the training and testing, while FakeEdge can both provably and empirically diminish the distribution gap.

TABLE 2.2

GIN'S PERFORMANCE IMPROVEMENT.

Layers	Cora	Citeseer	Pubmed	USAir	NS	PB	Yeast	C.ele	Power	Router	E.coli
1	↑2.80%	↑3.65%	↑4.53%	↑0.29%	↑1.30%	↑1.02%	↑1.54%	↑2.13%	↑5.24%	↑11.19%	↑1.67%
2	↑4.66%	↑14.53%	↑6.64%	↑0.73%	↑1.55%	↑2.16%	↑3.40%	↑5.41%	↑25.32%	↑14.73%	↑2.59%
3	↑9.78%	↑15.19%	↑6.57%	↑0.98%	↑2.49%	↑2.43%	↑3.60%	↑4.48%	↑20.46%	↑13.38%	↑3.14%

GIN's performance improvement by *Edge Att* compared to *Original* with a different number of layers. GIN utilizes mean-pooling as the subgraph-level readout.

2.6.5 Dataset shift with deeper GNNs

Given two graphs with n nodes in each graph, 1-WL test may take up to n iterations to determine whether two graphs are isomorphic [132]. Thus, GNNs, which mimic 1-WL test, tend to discriminate more non-isomorphic graphs when the number of GNN layers increases. SEAL [182] has empirically witnessed a stronger representation power and obtained more expressive link representation with deeper GNNs. However, we notice that the dataset shift issue in the subgraph link prediction becomes more severe when GNNs try to capture long-range information with more layers.

We reproduce the experiments on GIN by using $l = 1, 2, 3$ message passing layers and compare the model performance by AUC scores with and without FakeEdge. Here we only apply *Edge Att* as the FakeEdge technique. The relative AUC score improvement of *Edge Att* is reported, namely $(AUC_{EdgeAtt} - AUC_{Original})/AUC_{Original}$. The results are shown in Table 2.2. As we can observe, the relative performance improvement between *Edge Att* and *Original* becomes more significant with more layers, which indicates that the dataset shift issue can be potentially more critical when we seek deeper GNNs for greater predictive power.

To explain such a phenomenon, we hypothesize that GNNs with more layers will

involve more nodes in the subgraph, such that their computation graph is dependent on the existence of the edge at the focal node pair. For example, select a node v from the subgraph $\mathcal{G}^r_{i,j}$, which is at least l hops away from the focal node pair $\{i, j\}$, namely $l = min(d(i, v), d(j, v))$. If the GNN has only l layers, v will not include the edge (i, j) in its computation graph. But with a GNN with $l + 1$ layers, the edge (i, j) will affect v's computation graph. We leave the validation of the hypothesis to future work.

2.6.6 Heuristic methods with FakeEdge

FakeEdge, as a model-agnostic technique, not only has the capability of alleviating the dataset shift issue for GNN-based models, but also can tackle the problem for heuristics methods. The heuristics link predictors assign a score to each focal node pair, indicating the likelihood of forming a new edge. Some of the conventional heuristic link predictors, like Common Neighbor [91], Adamic–Adar index [4], or Resource Allocation [191], are Edge Invariant because these predictors are independent of the existence of the target link.

However, other link predictors, including Preferential Attachment (PA) [9] and Jaccard Index (Jac) [69], are not Edge Invariant. The existence/absence of the target link can change the values of the predictors, which in turn changes the ranking of focal node pairs. The original PA for a focal node pair i, j is $PA(i, j) = |\mathcal{N}(i)||\mathcal{N}(j)|$, where $\mathcal{N}(i)$ is the neighbors of node i. After applying *Edge Plus*, $PA^{plus}(i, j) = |\mathcal{N}(i) \cup \{j\}||\mathcal{N}(j) \cup \{i\}|$. Similarly, $Jac^{plus}(i, j) = |\mathcal{N}(i) \cap \mathcal{N}(j)|/|\mathcal{N}(i) \cup \mathcal{N}(j) \cup \{i, j\}|$.

We follow the same protocol in the previous experiment. As shown in Table 2.3, *Edge Plus* can significantly improve the performance of the PA predictor on several datasets. With FakeEdge, PA performs over 10% better on Pubmed. Surprisingly, even though PA is not able to predict the links on Router dataset with AUC score lower than 50%, PA with Edge Plus achieves 74% AUC score and becomes a functional

TABLE 2.3

FAKEEDGE ON HEURISTIC METHODS.

Models	Fake Edge	Cora	Citeseer	Pubmed	USAir	NS	PB	Yeast	C.ele	Power	Router	E.coli
PA	*Original*	63.15±1.38	58.20±2.18	71.72±0.36	88.84±1.41	66.19±1.82	90.05±0.52	82.10±1.15	75.72±2.20	44.47±1.58	48.20±0.83	91.99±0.78
	Edge Plus	**65.05±1.31**	**61.05±1.96**	**84.04±0.37**	**90.36±1.45**	65.29±1.97	**90.47±0.49**	**82.66±0.98**	**75.98±2.31**	46.83±1.61	**74.03±1.05**	91.98±0.78
Jac	*Original*	71.76±0.85	66.33±1.23	64.41±0.20	88.89±1.55	92.19±0.80	86.82±0.60	88.49±0.53	78.77±1.94	58.18±0.50	55.77±0.55	81.43±0.92
	Edge Plus	**71.77±0.85**	66.33±1.23	**64.42±0.20**	**89.65±1.45**	92.19±0.80	**87.20±0.58**	**88.52±0.53**	**79.33±1.88**	58.18±0.50	55.77±0.55	**81.79±0.90**

Heuristic methods with/without FakeEdge (AUC). The best results are highlighted in bold.

link predictor. In terms of Jac, we observe that Jac with FakeEdge can only gain marginal improvement. This is because that even though Jac is dependent on the existence of target link, the change of Jac index is relatively small when the existence of the target link flips.

2.7 Conclusion

Dataset shift is arguably one of the most challenging problems in the world of machine learning. However, to the best of our knowledge, none of the previous studies sheds light on this notable phenomenon in link prediction. In this paper, we studied the issue of dataset shift in link prediction tasks with GNN-based models. We first unified several existing models into a framework of subgraph link prediction. Then, we theoretically investigated the phenomenon of dataset shift in subgraph link prediction and proposed a model-agnostic technique FakeEdge to amend the issue. Experiments with different models over a wide range of datasets verified the effectiveness of FakeEdge.

CHAPTER 3

ROBUST GNNS BY DATA AUGMENTATION

3.1 Overview

Chapter 3 addresses the robustness challenges of GNNs from a data-oriented perspective by focusing on the quality and completeness of the underlying graph data. Although link prediction (LP) is a core task within graph representation learning, the presence of noisy or incorrect edges, as well as missing edges, can severely limit the generalization capability of LP models. To overcome these issues, we introduce COmplete and REduce (CORE), a novel data augmentation method inspired by the Information Bottleneck principle. CORE enhances the robustness of LP models by simultaneously identifying and removing spurious edges and recovering essential but missing links, thereby generating more reliable and predictive graph data. Extensive empirical evaluation on multiple benchmark datasets confirms the effectiveness of CORE, demonstrating its superiority over current methods.

This chapter primarily builds upon a pre-print manuscript [39] in collaboration with Zhichun Guo, and Nitesh V. Chawla.

3.2 Introduction

Graph-structured data is ubiquitous in various domains, including social networks [91], recommendation systems [85], and protein-protein interactions [141]. Link prediction (LP), the task of predicting missing or future edges in a graph, is a fundamental problem in graphs. Over the years, a plethora of link prediction algorithms

have been proposed, ranging from heuristics-based link predictors [4, 78, 91, 191] to more sophisticated graph neural network (GNN) based methods [81, 118, 178].

One major challenge in LP is the quality, reliability, and veracity of graph data. In many real-world scenarios, data collection can be difficult due to factors such as incomplete information, errors in data labeling, and noise introduced by measurement devices or human mistakes [26, 196]. As a consequence, the graph constructed based on the collected data may contain missing or erroneous edges. This can subsequently impact the performance of LP models. Moreover, the excessive dependence on noisy graphs can impede the model's capability in distinguishing the real and spurious edges, which harms the generalizability of models. Therefore, the question of preserving the robust learning capacity and generalizability of LP models on noisy graphs remains unresolved.

To mitigate the degradation of model performance on noisy data with inferior data quality, data augmentation (DA) has emerged as a powerful technique by artificially expanding the training dataset with transformed versions of the original data instances, primarily in the field of computer vision [89, 133]. However, in the context of LP, few works have been proposed to overcome the limitation of models on noisy graphs [186]. For example, CFLP [185] employs causal inference by complementing counterfactual links into the observed graph. Edge Proposal [134] seeks to inject highly potential edges into the graph as a signal-boosting preprocessing step. Nevertheless, these works fail to consider the inherent noise or that brought by the augmentation process, holding an implicit assumption that the observed graph truly reflects the underlying relationships

In this paper, we investigate how to augment the graph data for link prediction to accomplish two primary goals: ***eliminating noise inherent in the data and recovering missing information in graphs.*** To augment with robust, diverse, and noise-free data, we employ the Information Bottleneck (IB) principle [142, 143].

IB offers a framework for constraining the flow of information from input to output, enabling the acquisition of a maximally compressed representation while retaining its predictive relevance to the task at hand [6]. Learning such an effective representation is particularly appealing because it gives us a flexible DA pipeline where we can seamlessly integrate other DA techniques without concern for the introduction of extraneous noise they might bring.

Present work. We introduce COmplete and REduce (CORE), a novel data augmentation framework tailored for the link prediction task. CORE comprises of two distinct stages: the Complete stage and the Reduce stage. The Complete stage addresses the incompleteness of the graph by incorporating highly probable edges, resulting in a more comprehensive graph representation. One can plug in any link predictors that may be advantageous in recovering the graph's structural information, despite the possibility of introducing noisy edges. The Reduce stage, which is the crux of the proposed method, operates on the augmented graph generated by the Complete stage. It aims to shrink the edge set while preserving those critical to the link prediction task. In doing so, the Reduce stage effectively mitigates any misleading information either inherently or introduced during the Complete stage. By adhering to the IB principle, the Reduce stage yields a minimal yet sufficient graph structure that promotes more generalizable and robust link prediction performance.

However, unlike DA in images where transformations can be applied independently, modifications to a single node or edge in a graph inevitably impact the surrounding neighborhood. This arises from the interdependence of data instances within a graph. Under these conditions, applying a universal DA to different instances in a graph may be suboptimal, as a specific augmentation could benefit one link prediction while negatively affecting another. For example, in Figure 3.1, the inference of different links may favor adding different edges into their neighborhood.

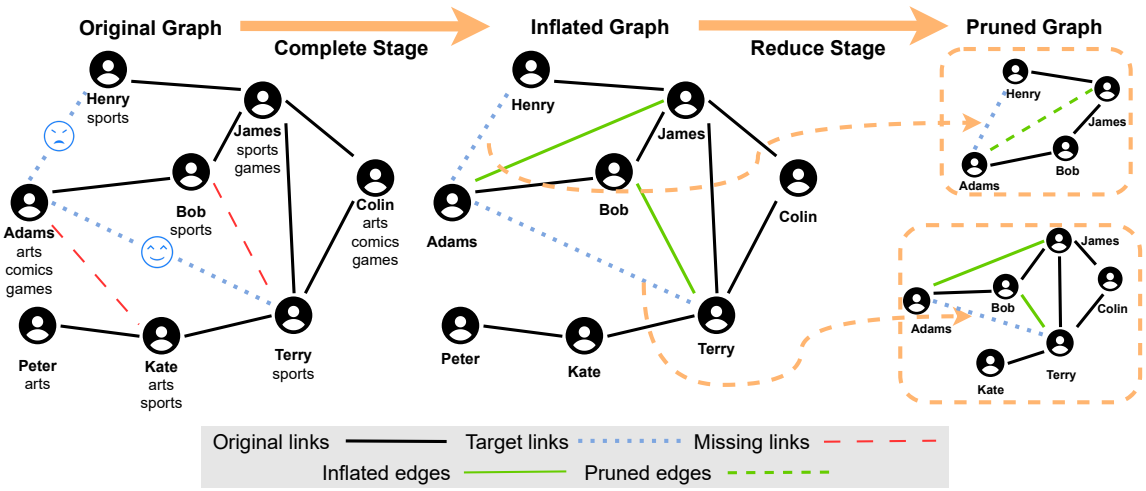

Figure 3.1: Overview of our CORE framework. It consists of two stages: (1) the Complete stage, which aims to recover missing edges by incorporating highly probable edges into the original graph, and (2) the Reduce stage, which is the core component of our method, designed to prune noisy edges from the graph in order to prevent overfitting on the intrinsic noise and that introduced by the Complete stage. Recognizing that predicting different links may require distinct augmentations, we extract the surrounding subgraph of each link and apply independent augmentations accordingly. In the social network example illustrated, assuming that Adams and Terry will become friends while Adams and Henry will not, tailored augmentations can facilitate more accurate link prediction by the model.

To address this dependency issue within a graph, we recast the link prediction task as a subgraph link prediction [37, 178]. In this context, we can apply different DAs to neighboring links without concerns about potential conflicts between their preferred augmentations. This approach allows for more targeted and effective augmentation, ultimately enhancing the performance of our CORE framework in link prediction tasks.

3.3 Preliminary

In this section, we introduce the notations and concepts utilized throughout the paper.

Graph and link prediction. Let $G = (V, E, \mathbf{X})$ represent an undirected graph, where V is the set of nodes with size n, indexed as $\{i\}_{i=1}^{n}$. $\mathcal{N}_v$ is the neighborhood of the node v. $E \subseteq V \times V$ denotes the observed set of edges, and $\mathbf{X}_i \in \mathcal{X}$ represents the feature of node i. The unobserved set of edges, denoted by $E_c \subseteq V \times V \setminus E$, comprises either missing edges or those expected to form in the future within the original graph G. Thus, for those links $(i, j) \in E \cup E_c$, we can assign label $Y = 1$ and regard them as positive samples, while the rest $\{(i, j) \subseteq V \times V | (i, j) \notin E \cup E_c\}$ we assign label $Y = 0$ as negatives. Based on the given graph G, the goal of the link prediction task is to compute the nodes similarity scores to identify the unobserved set of edges E_c [93]. Numerous heuristic models are proposed for link prediction task over time, including Common Neighbor (CN) [91], Adamic-Adar index (AA) [4], Resource Allocation (RA) [191], and Katz index [78]. While these traditional approaches effectively utilize the topological structure of the graph, GNN-based models [81, 178] exhibit a superior ability to exploit both the structure and node attributes associated with the graph.

Subgraph link prediction. Even though some link predictors, such as the Katz index and PageRank [15], require the entire graph to calculate similarity scores for a target link, many others only rely on a local neighborhood surrounding the target link for computation. For instance, the Common Neighbor predictor necessitates only a 1-hop neighborhood of the target link, and generally, an l-layer GNN requires the l-hop neighborhood of the target link. Moreover, Zhang and Chen [178] has demonstrated that local information can be sufficient for link prediction tasks. As a result, the link prediction task for a specific target link can be reformulated as a graph classification problem based on the local neighborhood of the target link, aiming to determine whether the link exists or not [37]. Formally, given a subgraph $G^{l}_{(i,j)}$ induced by the nodes l-hop reachable from node pair (i, j), a subgraph link prediction is a task of predicting the label $Y \in \{0, 1\}$ for the subgraph, where $Y = 1$

indicates that the target link exists, and vice versa.

Data augmentation. Data augmentation is the process of expanding the input data by either slightly perturbing existing data instances or creating plausible variations of the original data. This technique has been proven effective in mitigating overfitting issues during training, particularly in the fields of computer vision [30] and natural language processing [49]. In the realm of graph representation learning, several DA methods have been proposed [186] to address challenges such as over-smoothing [127], generalization [26], and over-squashing [145]. However, most graph data augmentation techniques have primarily focused on node and graph classification tasks, with relatively limited exploration in the context of LP [185].

3.4 Proposed framework: CORE

In this section, we present our proposed two-stage data augmentation framework for LP, referred to as CORE. We begin by introducing the Complete stage, which aims to recover missing edges in the original graph. Following this, we discuss the Reduce stage, the most critical component of our proposed method, designed to eliminate noisy and spurious edges in the graph. Finally, we outline a practical implementation that leverages the Graph Information Bottleneck (GIB) [163] for pruning inflated edges. The overview of the framework is shown in Figure 3.1.

3.4.1 Complete stage: inflating missing connections

The data collection process is inherently susceptible to errors, which can result in incomplete or even erroneous structural information in the original graph. Furthermore, the nature of the link prediction task involves identifying missing or potentially newly forming edges, implicitly assuming that graph data is incomplete. Therefore, mitigating the incompleteness of graph structures can be advantageous. In Theo-

rem 3, we will also see how inflating missing edges can help identify the most crucial component that determines whether a link should exist.

Implementation. We begin with a simple and straightforward method introduced by Singh et al. [134] to inflate the original graph with additional edges. Although more sophisticated graph completion methods [155] can also be plugged into the Complete stage, we find that employing a straightforward, low-computational-cost algorithm is sufficient for augmenting the graph structures effectively.

Due to the sparsity of most real-world graphs, the number of potentially missing edges is proportional to the quadratic of the number of nodes $\mathcal{O}(n^2)$. Scoring all non-connected node pairs can be computationally prohibitive. To reduce the size of the candidate node pairs, we only consider those non-connected pairs that have at least one common neighbor for potential addition to the graph. Subsequently, we can use any link prediction method to score these candidate node pairs. For large-scale datasets, like OGB-Collab [62], we can rely on the faster computation of non-parametric heuristic methods. For graphs of moderate size, one may choose any heuristic methods or GNN-based methods like GCN [82] and SAGE [59]. Note that only scoring node pairs with common neighbors is a pragmatic choice. Most real-world graphs, governed by popular models such as the assortative SBM [60] or the Watts–Strogatz model [158], exhibit higher connection likelihood for node pairs with shared neighbors. Therefore, this design balances computational efficiency with empirical effectiveness, a trade-off we believe to be minimal but crucial for practical purposes. An empirical investigation can be found in Appendix B.3.6.

After scoring all the candidate node pairs, we sort them based on their similarity scores and select the top k node pairs E_{ext} to add to the original graph, where k is a hyperparameter. Thus, the graph G becomes $G^+ = (V, E \cup E_{ext}, \mathbf{X})$. It is important to note that, although we add these predicted links to the graph, we mark them as

inflated edges to differentiate them from the original graph. These inflated edges will not be used as training signals for later stages but will only serve as a complement to the graph's topological structure. This distinction is crucial, as we can tolerate the noisy edges in the input space but do not want to introduce any noise to the labels of LP in our DA process.

3.4.2 Reduce stage: pruning noisy edges

The Reduce stage is the central aspect of our CORE data augmentation framework. Inspired by GSAT [103], we leverage GIB [163] to parameterize a reducer, which constrains the graph structure to a minimal yet sufficient graph component for link prediction. The resulting graph component is expected to achieve three goals: (1) remove task-irrelevant information from the data (the regularization in Equation 3.6); (2) prune the graph structures so that only the most predictive graph components remain for inference (the log-likelihood in Equation 3.6); and (3) provide diversified augmentation to the original data (the edge sampling step). We begin by introducing the necessity of decoupling the DA for each link. Then we discuss the GIB objective and its tractable variational bound. Finally, we present the implementation of our data augmentation in the Reduce stage.

Interdependence of graph data. In the link prediction task, the data instances we are interested in are the links in the graph. However, unlike images, links in a graph are correlated; the existence and properties of each link are dependent on one another. Consequently, when applying DA to a specific link, it will inevitably affect the environment of other links, especially those in close proximity. This can yield suboptimal results, as links will compete with each other to obtain the best augmentation for their own sake. Furthermore, it becomes computationally infeasible to apply the IB principle when the i.i.d assumption does not hold [163].

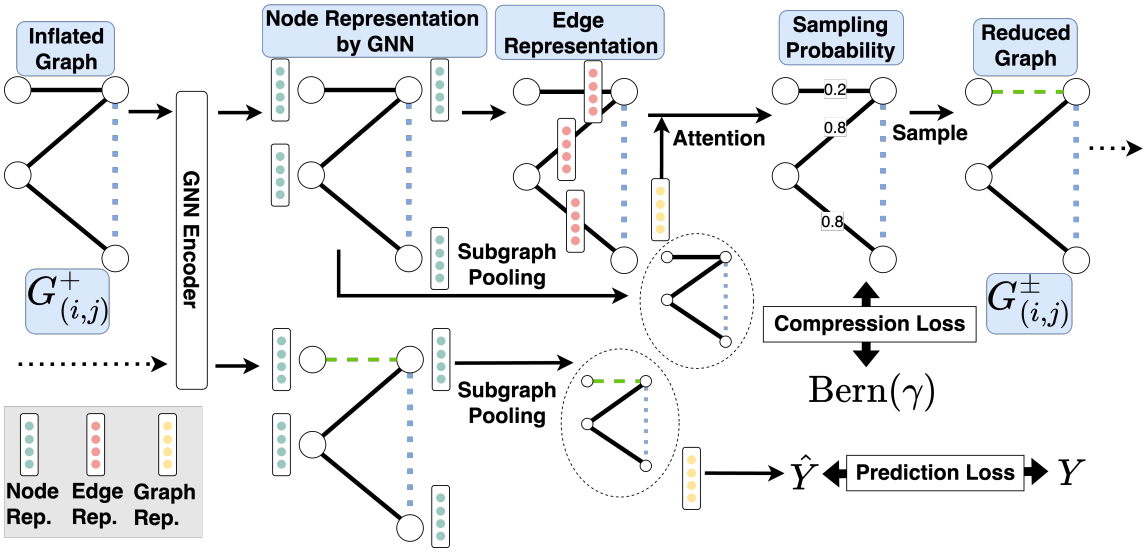

Figure 3.2: The Reduce stage commences with the inflated subgraph $G^{+}_{(i,j)}$ surrounding the target link (i, j). We first apply a GNN to encode node representations, followed by edge representation derived from the node encodings. To compute sampling probability scores for each edge, we utilize an attention mechanism that combines the edge representation with the subgraph pooling. Since the subgraph pooling encapsulates information from the entire subgraph and is employed for target link prediction, the generated probability scores reflect not only the edge's inherent property but also its relationship to the target link (i, j). Subsequently, we sample each edge using a Bernoulli distribution based on its probability to obtain the pruned graph. Finally, the pruned graph $G^{\pm}_{(i,j)}$ is fed back into the model as augmented input for enhanced graph structures.

To address these issues, we reformulate LP as a subgraph link prediction. Subgraph link prediction allows for decoupling the overlapping environments of each link, making it possible to have a different DA for each link. Specifically, for each node pair (i, j), we extract its l-hop enclosing subgraph $G^{+;l}_{(i,j)}$ from the entire graph G^{+}. To simplify notation when there is no ambiguity, we may omit the number of hops and represent the subgraph as $G^{+}_{(i,j)}$. It is worth noting that prior works also adopt a similar strategy to handle the non i.i.d nature of graph data [156, 187] when perturbing the data. In Section 3.5.3, we empirically examine the optimal DAs tailored to different target links. Notably, the DA derived from a single edge may vary depending

on the target link under consideration.

GIB. In general, IB aims to learn a concise representation Z from the input X that is also expressive for the output y, measured by the mutual information between the latent representation and input/output [6, 142]. Thus, the objective is:

$$\max_{Z} I(Z, Y) \text{ s.t. } I(X, Z) \leq I_c. \tag{3.1}$$

where $I(\cdot, \cdot)$ denotes the mutual information and I_c is the information constraint. In the context of LP, we can regard the enclosing subgraph $G^{+}_{(i,j)}$ as input X, including both the node attributes and graph structure. Y is the link's existence at (i, j), and Z is the latent representation.

While the original GIB [163] constrains the information flow from both node attributes of a graph and graph structures, we propose to only constrain the structural information in our data augmentation for the link prediction task. Compared to node attributes in a graph, graph structures are overwhelmingly more critical for the link prediction task [93, 118]. Moreover, many graphs without node attributes still exhibit the need for link prediction. Thus, we define our objective as:

$$\max_{G^{\pm}_{(i,j)} \in \mathbb{G}_{\text{sub}}(G^{+}_{(i,j)})} I(G^{\pm}_{(i,j)}, Y) \text{ s.t. } I(G^{\pm}_{(i,j)}, G^{+}_{(i,j)}) \leq I_c. \tag{3.2}$$

where $G^{\pm}_{(i,j)} \in \mathbb{G}_{\text{sub}}(G^{+}_{(i,j)})$ is a subgraph pruned from the inflated graph $G^{+}_{(i,j)}$. In other words, we aim to find the subgraph of the inflated graph that is simultaneously the most predictive and concise for the link prediction task. We assume that this graph reduction process can prune the noisy edges introduced by the previous Complete stage while retaining the beneficial added information. Our method shares a similar spirit with GSAT [103] and IB-subgraph [175], as we explore finding a subgraph structure that is most essential for the task.

Next, by introducing a Lagrange multiplier β, we obtain the unconstrained version of the objective:

$$\min_{G^{\pm}_{(i,j)} \in \mathbb{G}_{\text{sub}}(G^{+}_{(i,j)})} -I(G^{\pm}_{(i,j)}, Y) + \beta I(G^{\pm}_{(i,j)}, G^{+}_{(i,j)}). \tag{3.3}$$

where β is the hyperparameter to balance the tradeoff between predictive power and compression.

The computation of the mutual information term $I(\cdot, \cdot)$ is, in general, computationally intractable. To address this issue, we follow the works of Alemi et al. [6], Miao et al. [103], Wu et al. [163] to derive a tractable variational upper bound for Equation 3.3. The detailed derivation is provided in Appendix B.4. To approximate the first term $I(G^{\pm}_{(i,j)}, Y)$, we derive a variational lower bound. The lower bound can be formulated as:

$$I(G^{\pm}_{(i,j)}; Y) \geq \mathbb{E}[\log q_\theta(Y|G^{\pm}_{(i,j)})]. \tag{3.4}$$

Essentially, q_θ is the predictor of our model, which can be a parameterized GNN.

For the second term $I(G^{\pm}_{(i,j)}, G^{+}_{(i,j)})$, we derive an upper bound by introducing a variational approximation $r(G^{\pm}_{(i,j)})$ for the marginal distribution of $G^{\pm}_{(i,j)}$:

$$I(G^{\pm}_{(i,j)}, G^{+}_{(i,j)}) \leq \mathbb{E}[\text{KL}(p_\phi(G^{\pm}_{(i,j)}|G^{+}_{(i,j)})||r(G^{\pm}_{(i,j)})] \tag{3.5}$$

where p_ϕ is the reducer to prune noisy edges. Then we can put everything together and get the empirical loss to minimize:

$$\mathcal{L} \approx \frac{1}{|E|} \sum_{(i,j) \in E} [-\log q_\theta(Y|G^{\pm}_{(i,j)}) \tag{3.6}$$

$$+ \beta \text{KL}(p_\phi(G^{\pm}_{(i,j)}|G^{+}_{(i,j)})||r(G^{\pm}_{(i,j)}))]. \tag{3.7}$$

Next, we discuss how to parameterize the predictor q_θ and the reducer p_ϕ in the Reduce stage, as well as the choice of marginal distribution $r(G^{\pm}_{(i,j)})$.

3.4.3 Implementation of the Reduce stage.

The overall architecture of the Reduce stage is shown in Figure 3.2. It is important to note that both the predictor q_θ and the reducer p_ϕ utilize a GNN encoder to encode the graph representation for either prediction or graph pruning purposes. These two components can share a common GNN encoder with the same parameters, as the entire training of the Reduce stage is end-to-end. This shared encoder allows for more efficient learning and reduces the number of parameters required in the model.

Subgraph encoding. The Reduce stage begins with encoding the inflated graph $G^{+}_{(i,j)}$ using a GNN. We can choose any Message Passing Neural Network (MPNN) [57] as the instantiation of the GNN encoder. The MPNN can be described as follows:

$$\mathbf{m}_v^{(l)} = \text{AGG}\left(\{\mathbf{h}_u^{(l)}, \mathbf{h}_v^{(l)}, \forall u \in \mathcal{N}_v\}\right), \tag{3.8}$$

$$\mathbf{h}_v^{(l+1)} = \text{UPDATE}\left(\{\mathbf{h}_v^{(l)}, \mathbf{m}_v^{(l)}\}\right). \tag{3.9}$$

where a neighborhood aggregation function AGG$(\cdot)$ and an updating function UPDATE$(\cdot)$ are adopted in the t-th layer of a T-layer GNN. Consequently, $\{\mathbf{h}_v^{(T)} | v \in G^{+}_{(i,j)}\}$ represents the node embeddings learned by the GNN encoder.

A typical link prediction method, such as GAE [81] or SEAL [178], can make a prediction by pooling the node representations, namely $\mathbf{h}_{G^{+}_{(i,j)}} = \text{POOL}\left(\mathbf{h}_v^{(T)} | v \in G^{+}_{(i,j)}\right)$, where $\mathbf{h}_{G^{+}_{(i,j)}}$ is the final representation for the node pair (i, j). In our data augmentation approach, however, we first need to prune the noisy edges in order to obtain more concise graph structures.

Reduce by edge sampling. After encoding the node representations, we proceed to prune the noisy edges in the inflated graph. We first represent each edge (u, v) in the inflated graph $G^+_{(i,j)}$ by concatenating the node representations of the two end nodes and a trainable embedding indicating whether this edge comes from the original graph $G_{(i,j)}$ or the Complete Stage. Specifically, we obtain $\mathbf{h}_{(u,v)} = \left[\mathbf{h}_u; \mathbf{h}_v; \tilde{\mathbf{h}}_{(u,v)}\right]$, where $[\cdot;\cdot]$ is the concatenation operation. Appending $\tilde{\mathbf{h}}_{(u,v)}$ to the edge representation enables the model to be aware of whether the edges are originally in the graph or introduced by link predictors at the Complete stage.

In this setting, the edge representation $\mathbf{h}_{(u,v)}$ solely contains information about its structural role in the inflated graph. While structurally similar edges might influence distinct target node pairs differently, this representation does not convey information about how the edge (u, v) in the local subgraph $G^+_{(i,j)}$ affects the prediction of the target node pair (i, j). To make the edge representation directly interact with the downstream link prediction task, we further apply an attention mechanism [147, 149] and attend it to the overall representation of the entire subgraph to define its importance for link prediction. We compute this as follows:

$$a_{(u,v)} = Q_\phi(\mathbf{h}_{G^+_{(i,j)}})^T K_\phi(\mathbf{h}_{(u,v)}) \big/ \sqrt{F''}, \tag{3.10}$$

where Q_ϕ and K_ϕ are two MLPs and F'' is the output dimension of the MLPs.

Unlike GAT [149], which directly applies the attention scores as edge weights in each layer, we use these scores $a_{(u,v)}$ to sample the edges to diversify the views of the graph. For each edge (u, v) in $G^+_{(i,j)}$, we sample an edge mask from the Bernoulli distribution $\omega_{(u,v)} \sim \text{Bern}(\text{sigmoid}(a_{(u,v)}))$, which masks off unnecessary edges in the graph for LP. To ensure the gradient can flow through the stochastic node here, we utilize the Gumbel-Softmax trick [70, 96]. This procedure gives us a way to generate the reduced subgraph $G^\pm_{(i,j)}$ by the variational distribution $p_\phi(G^\pm_{(i,j)}|G^+_{(i,j)})$.

To control the marginal distribution in Equation 3.5, we follow [103, 163] and apply a non-informative prior $r(\tilde{G}_{(i,j)})$. In other words, $\tilde{G}_{(i,j)}$ is obtained by sampling edge connectivity $\tilde{\omega}_{(u,v)} \sim \text{Bern}(\gamma)$ for every node pair (u, v) in $G^{+}_{(i,j)}$. γ is a hyper-parameter. Regardless the graph structure of $G^{\pm}_{(i,j)}$, we connect (u, v) if $\tilde{\omega}_{(u,v)} = 1$ and disconnect the rest. This is essentially an Erdös-Rényi random graph [46]. The derivation of the KL loss term with respect to the marginal distribution is detailed in Appendix B.2.1.

Prediction based on pruned subgraph. Once we obtain the edge mask ω, we can encode the subgraph and make a link prediction. The edge mask can be regarded as the edge weight of the inflated graph. In this way, the edge weight plays the role of a message passing restrictor to prune the noisy edges in the inflated graph, which modifies the message passing part of Equation 3.8 as follows:

$$\mathbf{m}_v^{(l)} = \text{AGG}\left(\{\omega_{(u,v)} * \mathbf{h}_u^{(l)}, \mathbf{h}_v^{(l)}, \forall u \in \mathcal{N}_v\}\right). \tag{3.11}$$

Using the representation learned from the reduced subgraph $G^{\pm}_{(i,j)}$, we can feed them into a pooling layer plus an MLP to estimate Y. This models the distribution $q_\theta(Y|G^{\pm}_{(i,j)})$.

3.4.4 Theoretical analysis

In this section, we provide a theoretical foundation for the integration of the Complete and Reduce stages in our data augmentation approach for link prediction tasks.

Theorem 3. *Assume that:* (1) *The existence* Y *of a link* (i, j) *is solely determined by its local neighborhood* $G^{*}_{(i,j)}$ *in a way such that* $p(Y) = f(G^{*}_{(i,j)})$*, where* f *is a deterministic invertible function;* (2) *The inflated graph contains sufficient structures for*

prediction $G^*_{(i,j)} \in \mathbb{G}_{sub}(G^+_{(i,j)})$. *Then* $G^{\pm}_{(i,j)} = G^*_{(i,j)}$ *minimizes the objective in Equation 3.3.*

The first assumption in Theorem 3 is consistent with a widely accepted *local-dependence* assumption [163, 178] for graph-structured data. The second assumption highlights the importance of incorporating enough structural information into the graph in the Complete Stage prior to executing the reduce operation. Even though we assume that the link existence Y is causally determined by $G^*_{(i,j)}$, there still can be some other spurious correlations between Y and $G^+_{(i,j)}$. These correlations can be brought by the environments [8, 86, 162], and the shift of such correlations in the testing phase can cause performance degradation for LP models.

Theorem 3 implies that under mild assumptions, optimizing the objective in Equation 3.3 can help us uncover the most crucial component of the graph, which determines whether a link should exist. As a result, our approach enables the elimination of noisy and spurious edges, thereby enhancing the generalizability of link prediction models. The proof can be found in Appendix B.5.

3.5 Experiments

In this section, we present experimental results for our proposed method. We first assess the performance of CORE in comparison to various baseline DA techniques for the link prediction task. Then, we illustrate that heuristic link predictors can also benefit from the augmented graph structure by CORE. Furthermore, we demonstrate its robustness against adversarial edge perturbations. Further details of the experiments can be found in Appendix B.2.

3.5.1 Experimental setup

Baseline methods. We select three heuristic link predictors for non-GNN models: *CN* [110], *AA* [4], and *RA* [191]. For GNN models that exploit node-level representa-

TABLE 3.1

LINK PREDICTION PERFORMANCE.

Model Type	Models	C.ele	USAir	Yeast	Router	CS	Physics	Computers	Collab
Heuristics	*CN*	54.31±0.00	82.59±0.00	72.71±0.00	9.11±0.00	38.99±0.00	63.44±0.00	25.48±0.00	61.37±0.00
	AA	57.34±0.00	87.53±0.00	72.71±0.00	9.11±0.00	67.44±0.00	74.38±0.00	31.14±0.00	64.17±0.00
	RA	64.34±0.00	87.53±0.00	72.71±0.00	9.11±0.00	67.44±0.00	74.68±0.00	34.17±0.00	63.81±0.00
Network Embedding	*Node2Vec*	50.82±3.24	74.12±2.12	82.11±2.74	32.53±4.23	63.32±3.84	60.72±1.85	28.48±3.42	48.88±0.54
	DeepWalk	48.62±2.82	73.80±1.98	81.24±2.38	31.97±3.92	64.18±3.98	60.58±2.24	27.49±3.08	50.37±0.34
	LINE	52.40±2.02	74.82±3.40	82.45±2.75	34.39±3.86	63.96±2.83	61.90±1.93	27.52±2.98	53.91±0.00
GNNs	*GCN*	57.32±4.52	82.14±1.99	80.33±0.73	35.16±1.60	60.69±8.56	69.16±4.61	32.70±1.97	44.75±1.07
	SAGE	42.14±5.62	82.85±4.01	78.34±1.08	35.76±2.97	31.44±8.24	22.87±22.53	14.53±6.28	48.10±0.81
	SEAL	67.32±2.71	91.76±1.17	82.50±2.08	60.35±5.72	65.23±2.08	71.83±1.44	35.80±1.38	63.37±0.69
	ELPH	66.06±3.00	88.16±1.21	78.92±0.78	59.50±1.89	67.84±1.27	69.60±1.22	33.64±0.77	64.58±0.32
	NCNC	60.42±1.89	83.22±0.82	73.11±2.07	57.13±0.66	65.73±2.57	72.87±1.80	37.17±1.86	**65.97±1.03**
DAs	*Edge Proposal*	70.19±2.95	86.35±1.35	81.59±0.51	36.20±2.61	62.44±2.68	70.34±2.89	33.76±2.08	65.48±0.00
	CFLP	54.36±3.41	89.09±1.12	73.57±1.06	50.62±3.33	OOM	OOM	OOM	OOM
	Node Drop	68.76±2.77	90.79±1.40	81.45±3.10	61.76±5.72	64.80±2.52	70.51±1.87	35.94±2.30	62.57±0.96
	Edge Drop	66.92±4.29	92.12±0.96	81.92±1.94	59.66±7.18	67.27±1.64	72.52±1.88	36.91±0.94	63.20±0.88
Ours	*Complete Only*	72.10±1.70	91.84±1.23	82.70±2.20	63.18±4.01	67.06±1.01	71.83±1.44	35.80±1.38	63.57±0.48
	Reduce Only	70.22±3.69	92.35±0.95	84.22±1.58	65.40±2.27	67.79±1.50	74.73±2.12	37.88±1.10	64.24±0.60
	CORE	**76.34±1.65**	**93.14±1.09**	**84.67±1.13**	**65.64±1.28**	**69.67±1.36**	**74.73±2.12**	**37.88±1.10**	65.62±0.50
p-values		0.0001**	0.0394**	0.0096**	0.0105**	0.0060**	0.0486**	0.3126	-

Link prediction performance evaluated by Hits@50. The **best-performing** method is highlighted in bold, while the second-best performance is underlined. OOM means out of memory.

tion, we employ the two most widely used architectures: *GCN* [82] and *SAGE* [59]. For the link prediction utilizing edge-level representation, we choose *SEAL* [178], *ELPH* [22] and *NCNC* [155] as the baseline.

We select *Edge Proposal* [134] and *CFLP* [185], as two representative DA baselines with *GCN* as backbone. For *SEAL*, we evaluate two standard graph perturbation techniques [174], *Node Drop* [120] and *Edge Drop* [127]. Then, we present **our** results with *Complete Only*, *Reduce Only*, and the combined *CORE*. Details about these baseline models can be found in Appendix B.3.1.

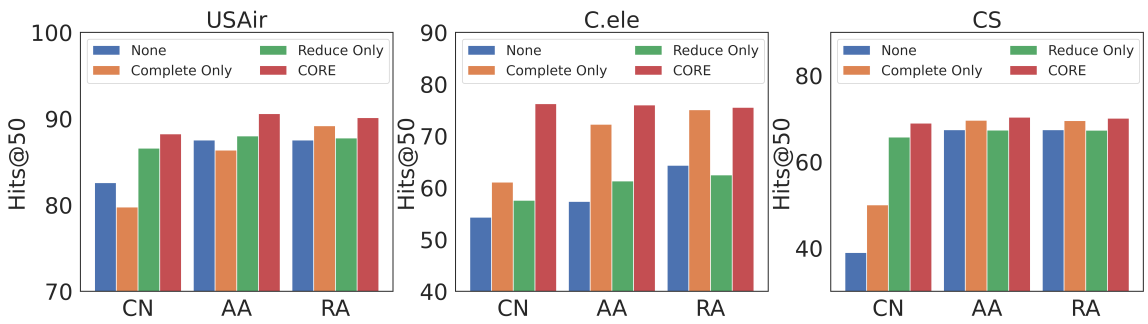

Figure 3.3: CORE can enhance the graph structure and even boost heuristics link predictors (Hits@50).

Benchmark datasets. We select four attributed and four non-attributed graphs as the benchmark. The attributed graphs consist of three collaboration networks, **CS**, **Physics** [131] and **Collab** [163], as well as a co-purchased graph, **Computers** [131]. The non-attributed graphs include **USAir** [10], **Yeast** [150], **C.ele** [158], and **Router** [136]. The comprehensive descriptions and statistics of the benchmark datasets can be found in Appendix B.3.2.

Evaluation protocols. We follow the evaluation settings from previous work [185] and split the links as 10% for validation, 20% for testing. For **Collab** [62], we use the official train-test split. The evaluation metric is Hits@50, which is widely accepted for evaluating link prediction tasks [62]. The results are reported for 10 different runs with varying model initializations.

3.5.2 Experimental results

Link prediction. Table 3.1 presents the link prediction performance of Hits@50 for all methods. Given the strong backbone model *SEAL*, we observe that our proposed data augmentation can further improve its performance on various datasets. In comparison to *SEAL* without any DA techniques, CORE consistently boosts the

performance by 1% to 9% in terms of Hits@50. More specifically, both *Complete Only* and *Reduce Only* can increase the model capability by different margins. Moreover, by combining those two stages together, CORE can almost always achieve the best performance and significantly outperforms baselines. Our results also reveal that CORE yields greater performance improvements when the available data size is limited. This observation suggests that models may be prone to overfitting to noise in low-data regimes. However, CORE effectively mitigates this issue by learning an underlying (Bernoulli) distribution associated with the graph structures, and prevents the model from overfitting to idiosyncratic structural perturbations.

Learnable and transferrable. One potential concern with using the reducer of CORE, which is a neural network possessing the capability of universal approximation [61], is that the performance improvement might be attributed to the overparameterization [11] of the model instead of the quality of our augmented graph. To address this concern and validate the efficacy of CORE as a DA method, we decouple the reducer from the model and investigate its ability to extract a generalizable view of the graph. We feed the augmented graph generated by the reducer to three heuristic link predictors: *CN*, *AA*, and *RA*. The results of this experiment can be found in Figure 3.3. Our findings demonstrate that the graph refined by CORE consistently improves the performance of heuristic link predictors. This outcome validates CORE's ability to learn a transferable and generalizable DA.

Robustness. To assess the robustness of our graph data augmentation method, we conduct additional experiments using an unsupervised graph poisoning attack, CLGA [183], to adversarially perturb the graph structures at varying attack rates. The results of this analysis can be found in Table 3.2 and Table B.2. Intriguingly, we observe that the advanced link prediction models, like *SEAL*, *ELPH* and *NCNC*,

exhibits a higher vulnerability to adversarial attacks compared to other baseline models. The capability to capture complex structural relationships of these expressive models renders them more sensitive to structural changes. However, our proposed method, CORE, leverages the robustness inherent in IB [6, 163] to enhance the model resilience by pruning spurious or even harmful perturbations. These findings suggest that the performance improvement offered by our method may be attributed to its ability to mitigate model vulnerability to adversarial perturbations.

***GIN* as backbone.** We examine whether CORE remains an effective DA technique when utilizing a different backbone model. In this case, we choose the Graph Isomorphism Network (*GIN*) [167], one of the most expressive GNNs, ensuring that the learned representation can encode structural information and guide downstream data augmentation. The results are presented in upper half of Table 3.3. We observe that, with *GIN* as the backbone, CORE can still improve link prediction performance over the baseline, yielding a 1% to 10% improvement. It indicates that CORE can be effectively integrated with other backbone models.

Information constraints and stochastic sampling. We further investigate the necessity of retaining the information constraint term in the objective function and the stochastic sampling component in the augmentation process. The results are displayed in the lower half of Table 3.3. By setting $\beta = 0$, the Reduce stage of our method loses the ability to constrain the information flow from the inflated graph. This leads to significant performance degradation, suggesting that the regularization term helps prevent the model from overfitting. Additionally, the performance declines when removing the stochastic sampling component and directly applying the attention score as the edge weight. This demonstrates that incorporating sampling in data augmentation can potentially expose the link prediction model to a wider

range of augmented data variations.

3.5.3 Different DAs for different target links

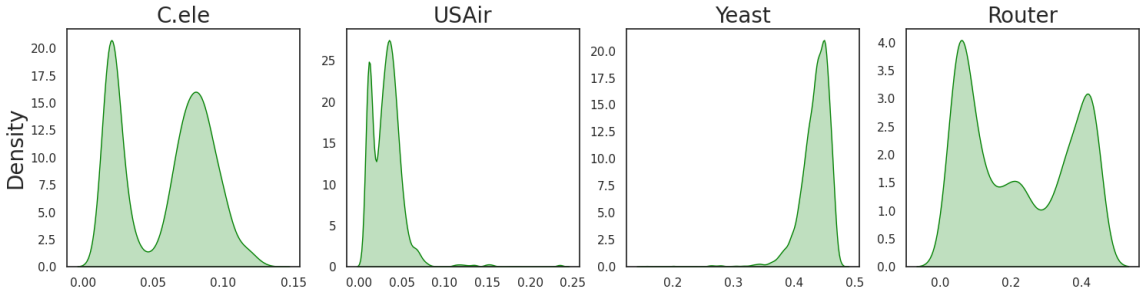

Figure 3.4: Histogram representing the standard deviations (std) of the learned edge mask ω for each edge within subgraphs associated with different target links. The frequent occurrence of larger std values implies substantial disagreement on the optimal DAs when focusing on different target links.

One of the unique designs of our methods is to augment each target link in a separate environment of its own. Here, we empirically investigate the necessity of isolating the DAs. We collect the edge mask ω for each edge within the subgraphs but across different target links. Then, for a set of such edge masks of the same edge, we calculate their standard deviations to indicate how much the learned edge masks ω agree or disagree with each other when augmenting different target links. The results are presented in Figure 3.4.

As it shows, while a portion of edges may have similar augmentation (small standard deviations), a significant part of them conflicts with each other (large standard deviations). On **C.ele** and **Router**, CORE will learn different DAs for nearly half of the edges associated with different target links. On **Yeast**, the majority of edges are

augmented differently by CORE. This result indicates that it is necessary to isolate the DA effect for each target link.

3.5.4 Additional ablation studies

To further substantiate the efficacy of our proposed DA method, we carry out extensive ablation studies. Due to page constraints, these detailed investigations are presented in the appendix. They include an analysis on the impact of different components of the Reduce stage (see Appendix B.3.3), a study on parameter sensitivity (see Appendix B.3.4), and evaluations of CORE when integrated with GCN and SAGE backbones (see Appendix B.3.5).

3.6 Conclusion

In this paper, we have introduced CORE, a novel data augmentation technique specifically designed for link prediction tasks. Leveraging the Information Bottleneck principle, CORE effectively eliminates noisy and spurious edges while recovering missing edges in the graph, thereby enhancing the generalizability of link prediction models. Our approach yields graph structures that reveal the fundamental relationships inherent in the graph. Extensive experiments on various benchmark datasets have demonstrated the effectiveness and superiority of CORE over competing methods, highlighting its potential as a leading approach for robust link prediction in graph representation learning.

TABLE 3.2

RESULTS OF ADVERSARIAL ROBUSTNESS.

Datasets	Methods	No Adv	10%	30%	50%
C.ele	*GCN*	$57.32_{\pm 4.52}$	$59.63_{\pm 3.41}$	$54.97_{\pm 3.08}$	$46.76_{\pm 3.90}$
	SAGE	$42.14_{\pm 5.62}$	$31.98_{\pm 6.26}$	$35.15_{\pm 3.38}$	$28.32_{\pm 5.74}$
	SEAL	$67.32_{\pm 2.71}$	$60.93_{\pm 2.23}$	$58.55_{\pm 1.46}$	$\underline{51.00_{\pm 2.32}}$
	ELPH	$66.06_{\pm 3.00}$	$62.28_{\pm 2.48}$	$56.62_{\pm 2.48}$	$50.40_{\pm 0.85}$
	NCNC	$60.42_{\pm 1.89}$	$52.10_{\pm 1.29}$	$53.40_{\pm 1.40}$	$50.26_{\pm 0.93}$
	Edge Proposal	$\underline{70.19_{\pm 2.95}}$	$\underline{64.71_{\pm 1.86}}$	$\underline{58.60_{\pm 2.14}}$	$50.93_{\pm 2.00}$
	CFLP	$54.36_{\pm 3.41}$	$51.75_{\pm 2.79}$	$46.49_{\pm 3.30}$	$42.83_{\pm 5.16}$
	CORE	$\mathbf{76.34_{\pm 1.65}}$	$\mathbf{72.03_{\pm 3.19}}$	$\mathbf{63.78_{\pm 2.24}}$	$\mathbf{58.16_{\pm 1.52}}$
USAir	*GCN*	$82.14_{\pm 1.99}$	$84.87_{\pm 1.22}$	$83.06_{\pm 1.73}$	$80.19_{\pm 0.77}$
	SAGE	$82.85_{\pm 4.01}$	$78.21_{\pm 2.81}$	$74.82_{\pm 3.28}$	$73.88_{\pm 3.65}$
	SEAL	$\underline{91.76_{\pm 1.17}}$	$85.51_{\pm 1.70}$	$84.80_{\pm 2.95}$	$81.53_{\pm 3.95}$
	ELPH	$88.16_{\pm 1.21}$	$\underline{86.71_{\pm 0.94}}$	$\underline{85.08_{\pm 0.96}}$	$\underline{84.54_{\pm 0.50}}$
	NCNC	$83.22_{\pm 0.82}$	$83.88_{\pm 0.78}$	$83.44_{\pm 0.50}$	$83.18_{\pm 0.53}$
	Edge Proposal	$86.35_{\pm 1.35}$	$86.42_{\pm 1.34}$	$84.95_{\pm 0.74}$	$80.94_{\pm 1.66}$
	CFLP	$89.09_{\pm 1.12}$	$86.53_{\pm 1.74}$	$77.26_{\pm 4.24}$	$80.32_{\pm 2.44}$
	CORE	$\mathbf{92.69_{\pm 0.75}}$	$\mathbf{89.72_{\pm 1.06}}$	$\mathbf{88.02_{\pm 1.13}}$	$\mathbf{86.71_{\pm 2.06}}$

Results of adversarial robustness for different models on *C.ele* and *USAir* datasets. The attack rates of 10%, 30%, and 50% represent the respective ratios of edges subjected to adversarial flips by CLGA [183].

TABLE 3.3

ABLATION STUDY.

Methods	**C.ele**	**USAir**	**Router**	**Yeast**
GIN as the backbone model				
GIN	$62.77_{\pm 2.33}$	$87.22_{\pm 2.70}$	$\underline{60.22_{\pm 2.09}}$	$\underline{75.38_{\pm 2.23}}$
Complete Only	$\underline{71.03_{\pm 2.18}}$	$\underline{88.12_{\pm 1.47}}$	$60.22_{\pm 2.09}$	$75.38_{\pm 2.23}$
Reduce Only	$64.13_{\pm 2.84}$	$88.71_{\pm 1.60}$	$62.81_{\pm 2.46}$	$78.40_{\pm 1.34}$
CORE	$\mathbf{72.33_{\pm 2.62}}$	$\mathbf{88.71_{\pm 1.60}}$	$\mathbf{62.81_{\pm 2.46}}$	$\mathbf{78.40_{\pm 1.34}}$
CORE without sampling or info constraint				
NoSample	$\underline{75.20_{\pm 1.71}}$	$91.51_{\pm 1.65}$	$64.35_{\pm 2.49}$	$\mathbf{84.59_{\pm 1.16}}$
$\beta = 0$	$74.02_{\pm 2.45}$	$\underline{91.81_{\pm 1.53}}$	$63.90_{\pm 2.16}$	$83.33_{\pm 2.05}$
NoSample-$\beta = 0$	$73.48_{\pm 2.52}$	$91.45_{\pm 1.73}$	$\underline{65.09_{\pm 1.41}}$	$\underline{84.47_{\pm 1.49}}$
CORE	$\mathbf{76.34_{\pm 1.65}}$	$\mathbf{92.69_{\pm 0.75}}$	$\mathbf{65.47_{\pm 2.44}}$	$84.22_{\pm 1.58}$

The upper half of the table presents results for CORE with GIN as the backbone model, while the bottom half investigates the impact of the balancing hyperparameter β and edge sampling in the proposed framework.

PART II

EFFICIENT GNNS

CHAPTER 4

EFFICIENT LINK-LEVEL REPRESENTATION WITH NODE-LEVEL COMPLEXITY

4.1 Overview

This chapter addresses the efficiency of GNNs by introducing a novel link prediction model that generates link-level structural features with only node-level computational complexity. Existing GNNs are known to be theoretically limited in their ability to capture certain critical graph substructures, restricting their expressive capability. For example, the widely-used Common Neighbor (CN) heuristic cannot be effectively modeled by standard GNN architectures due to their inherent limitations. Although explicitly extracting such structural information can enhance model performance, it involves quadratic computational costs, making it impractical for large-scale applications. To overcome this, we demonstrate that, by leveraging the orthogonality of input vectors, a pure message-passing mechanism can implicitly capture essential link-level structural features like CN. Motivated by this insight, we propose the Message Passing Link Predictor (MPLP), a method capable of approximating link-level substructures efficiently, maintaining node-level complexity. Experiments conducted on diverse benchmarks illustrate that MPLP consistently achieves state-of-the-art performance, highlighting its capability to efficiently approximate critical graph substructures.

This chapter primarily builds upon our collaborative study previously published in [38] with Zhichun Guo and Nitesh V. Chawla.

4.2 Introduction

Link prediction is a cornerstone task in the field of graph machine learning, with broad-ranging implications across numerous industrial applications. From identifying potential new acquaintances on social networks [91] to predicting protein interactions [141], from enhancing recommendation systems [85] to completing knowledge graphs [195], the impact of link prediction is felt across diverse domains. Recently, with the advent of Graph Neural Networks (GNNs) [82] and more specifically, Message-Passing Neural Networks (MPNNs) [57], these models have become the primary tools for tackling link prediction tasks. Despite the resounding success of MPNNs in the realm of node and graph classification tasks [59, 82, 149, 167], it is intriguing to note that their performance in link prediction does not always surpass that of simpler heuristic methods [62].

[182] highlights the limitations of GNNs/MPNNs for link prediction tasks arising from its intrinsic property of permutation invariance. Owing to this property, isomorphic nodes invariably receive identical representations. This poses a challenge when attempting to distinguish links whose endpoints are isomorphic nodes. As illustrated in Figure 4.1, nodes v_1 and v_3 share a Common Neighbor v_2, while nodes v_1 and v_5 do not. Ideally, due to their disparate local structures, these two links (v_1, v_3) and (v_1, v_5) should receive distinct predictions. However, the permutation invariance of MPNNs results in identical representations for nodes v_3 and v_5, leading to identical predictions for the two links. As [182] asserts, such node-level representation, even with the most expressive MPNNs, **cannot** capture structural link representation such as Common Neighbors (CN), a critical aspect of link prediction.

In this work, we posit that the pure Message Passing paradigm [57] can indeed capture structural link representation by exploiting orthogonality within the vector space. We begin by presenting a motivating example, considering a non-attributed graph as depicted in Figure 4.1. In order to fulfill the Message Passing's requirement

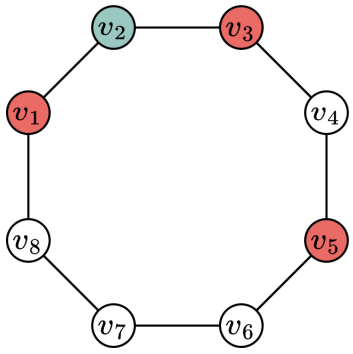

Figure 4.1: Isomorphic nodes result in identical MPNN node representation, making it impossible to distinguish links such as (v_1, v_3) and (v_1, v_5) based on these representations.

for node vectors as input, we assign a one-hot vector to each node v_i, such that the i-th dimension has a value of one, with the rest set to zero. These vectors, viewed as *signatures* rather than mere permutation-invariant node representations, can illuminate pairwise relationships. Subsequently, we execute a single iteration of message passing as shown in Figure 4.2, updating each node's vector by summing the vector of its neighbors. This process enables us to compute CN for any node pair by taking the inner product of the vectors of the two target nodes.

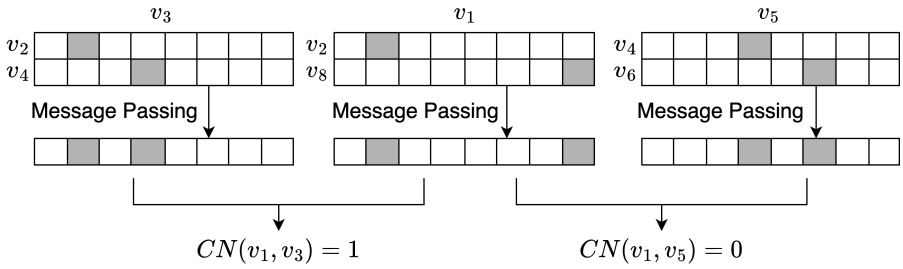

Figure 4.2: MPNN counts Common Neighbor through the inner product of neighboring nodes' one-hot representation.

At its core, this naive method employs an orthonormal basis as the node signatures, thereby ensuring that the inner product of distinct nodes' signatures is consistently zero. While this approach effectively computes CN, its scalability poses a significant challenge, given that its space complexity is quadratically proportional to the size of the graph. To overcome this, we draw inspiration from DotHash [114] and capitalize on the premise that the family of vectors almost orthogonal to each other swells exponentially, even with just linearly scaled dimensions [76]. Instead of relying on the orthogonal basis, we can propagate these quasi-orthogonal (QO) vectors and utilize the inner product to estimate the joint structural information of any node pair.

In sum, our paper presents several pioneering advances in the realm of GNNs for link prediction:

- We are the first, both empirically and theoretically, to delve into the proficiency of GNNs in approximating heuristic predictors like CN for link prediction. This uncovers a previously uncharted territory in GNN research.
- Drawing upon the insights gleaned from GNNs' capabilities in counting CN, we introduce **MPLP**, a novel link prediction model. Uniquely, MPLP discerns joint structures of links and their associated substructures within a graph, setting a new paradigm in the field.
- Our empirical investigations provide compelling evidence of MPLP's dominance. Benchmark tests reveal that MPLP not only holds its own but outstrips state-of-the-art models in link prediction performance.

4.3 Preliminaries and Related Work

Notations. Consider an undirected graph $G = (V, E, \boldsymbol{X})$, where V represents the set of nodes with cardinality n, indexed as $\{1, \ldots, n\}$, $E \subseteq V \times V$ denotes the observed set of edges, and $\boldsymbol{X}_{i,:} \in \mathbb{R}^{F_x}$ encapsulates the attributes associated with node i. Additionally, let $\mathcal{N}_v$ signify the neighborhood of a node v, that is $\mathcal{N}_v = \{u|\text{SPD}(u, v) = 1\}$ where the function $\text{SPD}(\cdot, \cdot)$ measures the shortest path distance

between two nodes. Furthermore, the node degree of v is given by $d_v = |\mathcal{N}_v|$. To generalize, we introduce the shortest path neighborhood $\mathcal{N}_v^s$, representing the set of nodes that are s hops away from node v, defined as $\mathcal{N}_v^s = \{u|\text{SPD}(u, v) = s\}$.

Link predictions. Alongside the observed set of edges E, there exists an unobserved set of edges, which we denote as $E_c \subseteq V \times V \setminus E$. This unobserved set encompasses edges that are either absent from the original observation or are anticipated to materialize in the future within the graph G. Consequently, we can formulate the link prediction task as discerning the unobserved set of edges E_c. Heuristics link predictors include Common Neighbor (CN) [91], Adamic-Adar index (AA) [4], and Resource Allocation (RA) [191]. CN is simply counting the cardinality of the common neighbors, while AA and RA count them weighted to reflect their relative importance as a common neighbor.

$$\begin{aligned} \text{CN}(u, v) &= \sum_{k \in \mathcal{N}_u \bigcap \mathcal{N}_v} 1; \\ \text{AA}(u, v) &= \sum_{k \in \mathcal{N}_u \bigcap \mathcal{N}_v} \frac{1}{\log d_k}; \\ \text{RA}(u, v) &= \sum_{k \in \mathcal{N}_u \bigcap \mathcal{N}_v} \frac{1}{d_k}. \end{aligned} \tag{4.1}$$

GNNs for link prediction. The advent of graphs incorporating node attributes has caused a significant shift in research focus toward methods grounded in GNNs. Most practical GNNs follow the paradigm of the Message Passing [57]. It can be formulated as:

$$\begin{aligned} \boldsymbol{m}_v^{(l)} &= \text{AGGREGATE}\left(\{\boldsymbol{h}_v^{(l)}, \boldsymbol{h}_u^{(l)}, \forall u \in \mathcal{N}_v\}\right), \\ \boldsymbol{h}_v^{(l+1)} &= \text{UPDATE}\left(\{\boldsymbol{h}_v^{(l)}, \boldsymbol{m}_v^{(l)}\}\right), \end{aligned} \tag{4.2}$$

where $\boldsymbol{h}_v^{(l)}$ represents the vector of node v at layer l and $\boldsymbol{h}_v^{(0)} = \boldsymbol{X}_{v,:}$. For simplicity, we use $\boldsymbol{h}_v$ to represent the node vector at the last layer. The specific choice of the neighborhood aggregation function, AGGREGATE$(\cdot)$, and the updating function, UPDATE$(\cdot)$, dictates the instantiation of the GNN model, with different choices leading to variations of model architectures. In the context of link prediction tasks, the GAE model [81] derives link representation, $\boldsymbol{h}(i, j)$, as a Hadamard product of the target node pair representations, $\boldsymbol{h}_{(i,j)} = \boldsymbol{h}_i \odot \boldsymbol{h}_j$.

Amplifying GNN Expressiveness with Randomness. The expressiveness of GNNs, particularly those of the MPNNs, has been the subject of rigorous exploration [167]. A known limitation of MPNNs, their equivalence to the 1-Weisfeiler-Lehman test, often results in indistinguishable representation for non-isomorphic graphs. A suite of contributions has surfaced to boost GNN expressiveness, of which [54, 100, 108, 179] stand out. An elegant, yet effective paradigm involves symmetry-breaking through stochasticity injection [1, 120, 129]. Although enhancing expressiveness, such random perturbations can occasionally undermine generalizability. Diverging from these approaches, our methodology exploits probabilistic orthogonality within random vectors, culminating in a robust structural feature estimator that introduces minimal estimator variance.

Link-Level Link Prediction. While node-level models like GAE offer enviable efficiency, they occasionally fall short in performance when compared with rudimentary heuristics [22]. To improve the capability of modeling link-level structures, the seminal SEAL model [178] labels nodes based on proximity to target links and then performs message-passing for each target link. However, it is hindered by computational expense, limiting its scalability. Efforts to build scalable link-level alternatives have culminated in innovative methods such as Neo-GNN [176], which distills struc-

tural features from adjacency matrices for link prediction. Elsewhere, ELPH [22] harnesses hashing mechanisms for structural feature representation, while NCNC [155] adeptly aggregates common neighbors' node representation. Notably, DotHash [114], which profoundly influenced our approach, employs quasi-orthogonal random vectors for set similarity computations, applying these in link prediction tasks.

Distinctively, our proposition builds upon, yet diversifies from, the frameworks of ELPH and DotHash. While resonating with ELPH's architectural spirit, we utilize a streamlined, efficacious hashing technique over MinHash for set similarity computations. Moreover, we resolve ELPH's limitations through strategic implementations like shortcut removal and norm rescaling. When paralleled with DotHash, our approach magnifies its potential, integrating it with GNNs for link predictions and extrapolating its applicability to multi-hop scenarios. It also judiciously optimizes variance induced by the structural feature estimator in sync with graph data. We further explore the potential of achieving higher expressiveness with linear computational complexity by estimating the substructure counting [27].

4.4 Can Message Passing count Common Neighbor?

In this section, we delve deep into the potential of MPNNs for heuristic link predictor estimation. We commence with an empirical evaluation to recognize the proficiency of MPNNs in approximating link predictors. Following this, we unravel the intrinsic characteristics of 1-layer MPNNs, shedding light on their propensity to act as biased estimators for heuristic link predictors and proposing an unbiased alternative. Ultimately, we cast light on how successive rounds of message passing can estimate the number of walks connecting a target node pair with other nodes in the graph. All proofs are provided in Appendix C.6.

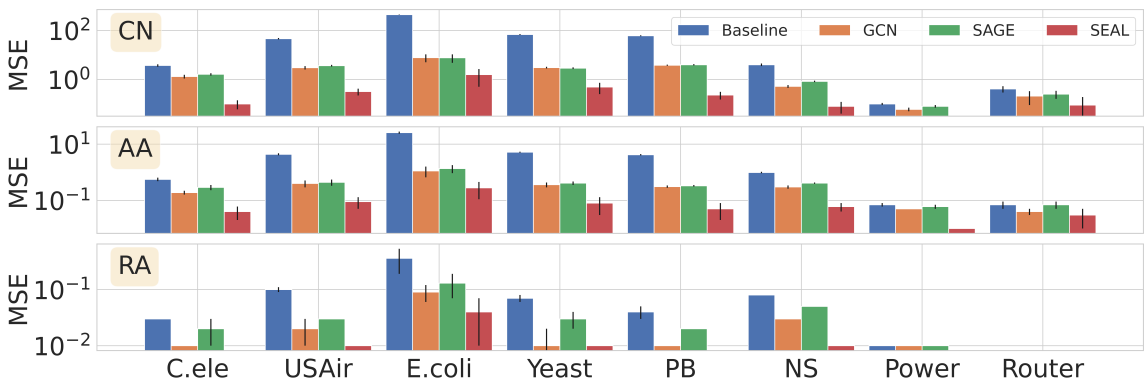

Figure 4.3: GNNs estimate CN, AA and RA via MSE regression, using the mean value as a Baseline. Lower values are better.

4.4.1 Estimation via Mean Squared Error Regression

To explore the capacity of MPNNs in capturing the overlap information inherent in heuristic link predictors, such as CN, AA and RA, we conduct an empirical investigation, adopting the GAE framework [81] with GCN [82] and SAGE [59] as representative encoders. SEAL [178], known for its proven proficiency in capturing heuristic link predictors, serves as a benchmark in our comparison. Additionally, we select a non-informative baseline estimation, simply using the mean of the heuristic link predictors on the training sets. The datasets comprise eight non-attributed graphs (more details in Section 4.6). Given that GNN encoders require node features for initial representation, we have to generate such features for our non-attributed graphs. We achieved this by sampling from a high-dimensional Gaussian distribution with a mean of 0 and standard deviation of 1. Although one-hot encoding is frequently employed for feature initialization on non-attributed graphs, we choose to forgo this approach due to the associated time and space complexity.

To evaluate the ability of GNNs to estimate CN information, we adopt a training procedure analogous to a conventional link prediction task. However, we reframe the task as a regression problem aimed at predicting heuristic link predictors, rather than

a binary classification problem predicting link existence. This shift requires changing the objective function from cross-entropy to Mean Squared Error (MSE). Such an approach allows us to directly observe GNNs' capacity to approximate heuristic link predictors.

Our experimental findings, depicted in Figure 4.3, reveal that GCN and SAGE both display an ability to estimate heuristic link predictors, albeit to varying degrees, in contrast to the non-informative baseline estimation. More specifically, GCN demonstrates a pronounced aptitude for estimating RA and nearly matches the performance of SEAL on datasets such as C.ele, Yeast, and PB. Nonetheless, both GCN and SAGE substantially lag behind SEAL in approximating CN and AA. In the subsequent section, we delve deeper into the elements within the GNN models that facilitate this approximation of link predictors while also identifying factors that impede their accuracy.

4.4.2 Estimation capabilities of GNNs for link predictors

GNNs exhibit the capability of estimating link predictors. In this section, we aim to uncover the mechanisms behind these estimations, hoping to offer insights that could guide the development of more precise and efficient methods for link prediction. We commence with the following theorem:

Theorem 4. *Let $G = (V, E)$ be a non-attributed graph and consider a 1-layer GCN/SAGE. Define the input vectors $\boldsymbol{X} \in \mathbb{R}^{N \times F}$ initialized randomly from a zero-mean distribution with standard deviation σ_{node}. Additionally, let the weight matrix $\boldsymbol{W} \in \mathbb{R}^{F' \times F}$ be initialized from a zero-mean distribution with standard deviation σ_{weight}. After performing message passing, for any pair of nodes $\{(u, v) | (u, v) \in$*

$V \times V \setminus E\}$, *the expected value of their inner product is given by:*

$$\text{GCN:}\ \ \mathbb{E}(\boldsymbol{h}_u \cdot \boldsymbol{h}_v) = \frac{C}{\sqrt{\hat{d}_u \hat{d}_v}} \sum_{k \in \mathcal{N}_u \bigcap \mathcal{N}_v} \frac{1}{\hat{d}_k};$$
$$\text{SAGE:}\ \ \mathbb{E}(\boldsymbol{h}_u \cdot \boldsymbol{h}_v) = \frac{C}{\sqrt{d_u d_v}} \sum_{k \in \mathcal{N}_u \bigcap \mathcal{N}_v} 1,$$

where $\hat{d}_v = d_v + 1$ *and* $C = \sigma^2_{node}\sigma^2_{weight}FF'$.

The theorem suggests that given proper initialization of input vectors and weight matrices, MPNN-based models, such as GCN and SAGE, can adeptly approximate heuristic link predictors. This makes them apt for encapsulating joint structural features of any node pair. Interestingly, SAGE predominantly functions as a CN estimator, whereas the aggregation function in GCN grants it the ability to weigh the count of common neighbors in a way similar to RA. This particular trait of GCN is evidenced by its enhanced approximation of RA, as depicted in Figure 4.3.

Quasi-orthogonal vectors. The GNN's capability to approximate heuristic link predictors is primarily grounded in the properties of their input vectors in a linear space. When vectors are sampled from a high-dimensional linear space, they tend to be quasi-orthogonal, implying that their inner product is nearly 0 w.h.p. With message-passing, these QO vectors propagate through the graph, yielding in a linear combination of QO vectors at each node. The inner product between pairs of QO vector sets essentially echoes the norms of shared vectors while nullifying the rest. Such a trait enables GNNs to estimate CN through message-passing. A key advantage of QO vectors, especially when compared with orthonormal basis, is their computational efficiency. For a modest linear increment in space dimensions, the number of QO vectors can grow exponentially, given an acceptable margin of error [76]. An intriguing observation is that the orthogonality of QO vectors remains intact even after GNNs undergo linear transformations post message-passing, attributed to the ran-

domized weight matrix initialization. This mirrors the dimension reduction observed in random projection [73].

Limitations. While GNNs manifest a marked ability in estimating heuristic link predictors, they are not unbiased estimators and can be influenced by factors such as node pair degrees, thereby compromising their accuracy. Another challenge when employing such MPNNs is their limited generalization to unseen nodes. The neural networks, exposed to randomly generated vectors, may struggle to transform newly added nodes in the graph with novel random vectors. This practice also violates the permutation-invariance principle of GNNs when utilizing random vectors as node representation. It could strengthen generalizability if we regard these randomly generated vectors as signatures of the nodes, instead of their node features, and circumvent the use of MLPs for them.

Unbiased estimator. Addressing the biased element in Theorem 4, we propose the subsequent instantiation for the message-passing functions:

$$\boldsymbol{h}_v^{(l+1)} = \sum_{u \in \mathcal{N}_v} \boldsymbol{h}_u^{(l)}. \tag{4.3}$$

Such an implementation aligns with the SAGE model that employs sum aggregation devoid of self-node propagation. This methodology also finds mention in DotHash [114], serving as a cornerstone for our research. With this kind of message-passing design, the inner product of any node pair signatures can estimate CN impartially:

Theorem 5. *Let $G = (V, E)$ be a graph, and let the vector dimension be given by $F \in \mathbb{N}_+$. Define the input vectors $\boldsymbol{X} = (X_{i,j})$, which are initialized from a random variable* x *having a mean of* 0 *and a standard deviation of $\frac{1}{\sqrt{F}}$. Using the 1-layer*

message-passing in Equation 4.3, for any pair of nodes $\{(u,v)|(u,v) \in V \times V\}$*, the expected value and variance of their inner product are:*

$$\mathbb{E}(\boldsymbol{h}_u \cdot \boldsymbol{h}_v) = \mathrm{CN}(u,v),$$
$$\mathrm{Var}(\boldsymbol{h}_u \cdot \boldsymbol{h}_v) = \frac{1}{F}\left(d_u d_v + \mathrm{CN}(u,v)^2 - 2\mathrm{CN}(u,v)\right) + F\mathrm{Var}\left(\mathrm{x}^2\right)\mathrm{CN}(u,v).$$

Though this estimator provides an unbiased estimate for CN, its accuracy can be affected by its variance. Specifically, DotHash recommends selecting a distribution for input vector sampling from vertices of a hypercube with unit length, which curtails variance given that $\mathrm{Var}(\mathrm{x}^2) = 0$. However, the variance influenced by the graph structure isn't adequately addressed, and this issue will be delved into in Section 4.5.

Orthogonal node attributes. Both Theorem 4 and Theorem 5 underscore the significance of quasi orthogonality in input vectors, enabling message-passing to efficiently count CN. Intriguingly, in most attributed graphs, node attributes, often represented as bag-of-words [122], exhibit inherent orthogonality. This brings forth a critical question: In the context of link prediction, do GNNs primarily approximate neighborhood overlap, sidelining the intrinsic value of node attributes? We earmark this pivotal question for in-depth empirical exploration in Appendix C.4, where we find that random vectors as input to GNNs can catch up with or even outperform node attributes.

4.4.3 Multi-layer message passing

Theorem 5 elucidates the estimation of CN based on a single iteration of message passing. This section explores the implications of multiple message-passing iterations and the properties inherent to the iteratively updated node signatures. We begin with a theorem delineating the expected value of the inner product for two nodes'

signatures derived from any iteration of message passing:

Theorem 6. *Under the conditions defined in Theorem 5, let* $\boldsymbol{h}_u^{(l)}$ *denote the vector for node* u *after the* l*-th message-passing iteration. We have:*

$$\mathbb{E}\big(\boldsymbol{h}_u^{(p)} \cdot \boldsymbol{h}_v^{(q)}\big) = \sum_{k \in V} |walks^{(p)}(k,u)||walks^{(q)}(k,v)|,$$

where $|walks^{(l)}(u,v)|$ *counts the number of length-*l *walks between nodes* u *and* v*.*

This theorem posits that the message-passing procedure computes the number of walks between the target node pair and all other nodes. In essence, each message-passing trajectory mirrors the path of the corresponding walk. As such, $\boldsymbol{h}_u^{(l)}$ aggregates the initial QO vectors originating from nodes reachable by length-l walks from node u. In instances where multiple length-l walks connect node k to u, the associated QO vector $\boldsymbol{X}_{k,:}$ is incorporated into the sum $|\text{walks}^{(l)}(k,u)|$ times.

One might surmise a paradox, given that message-passing calculates the number of walks, not nodes. However, in a simple graph devoid of self-loops, where at most one edge can connect any two nodes, it is guaranteed that $|\text{walks}^{(1)}(u,v)| = 1$ iff $\text{SPD}(u,v) = 1$. Consequently, the quantity of length-1 walks to a target node pair equates to CN, a first-order heuristic. It's essential to recognize, however, that $|\text{walks}^{(l)}(u,v)| \geq 1$ only implies $\text{SPD}(u,v) \leq l$. This understanding becomes vital when employing message-passing for estimating the local structure of a target node pair in Section 4.5.

4.5 Method

In this section, we introduce our novel link prediction model, denoted as **MPLP**. Distinctively designed, MPLP leverages the pure essence of the message-passing mechanism to adeptly learn joint structural features of the target node pairs.

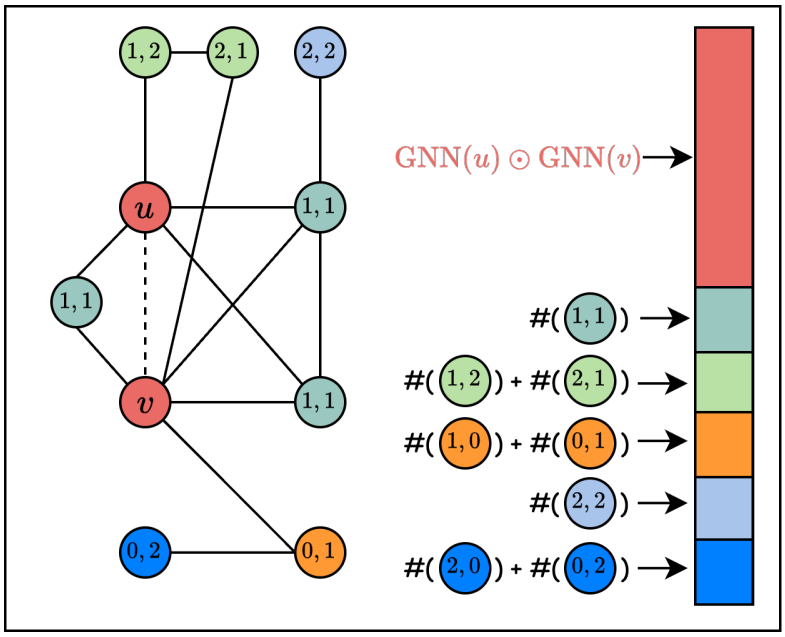

Figure 4.4: Representation of the target link (u, v) within our model (MPLP), with nodes color-coded based on their distance from the target link.

Node representation. While MPLP is specifically designed for its exceptional structural capture, it also embraces the inherent attribute associations of graphs that speak volumes about individual node characteristics. To fuse the attributes (if they exist in the graph) and structures, MPLP begins with a GNN, utilized to encode node u's representation: $\text{GNN}(u) \in \mathbb{R}^{F_x}$. This node representation will be integrated into the structural features when constructing the QO vectors. Importantly, this encoding remains flexible, permitting the choice of any node-level GNN.

4.5.1 QO vectors construction

Probabilistic hypercube sampling. Though deterministic avenues for QO vector construction are documented [74, 75], our preference leans toward probabilistic techniques for their inherent simplicity. We inherit the sampling paradigm from DotHash [114], where each node k is assigned with a node signature $\boldsymbol{h}_k^{(0)}$, acquired via random sampling from the vertices of an F-dimensional hypercube with unit vec-

tor norms. Consequently, the sampling space for $\boldsymbol{h}_k^{(0)}$ becomes $\{-1/\sqrt{F}, 1/\sqrt{F}\}^F$.

Harnessing One-hot hubs for variance reduction. The stochastic nature of our estimator brings along an inevitable accompaniment: variance. Theorem 5 elucidates that a graph's topology can augment estimator variance, irrespective of the chosen QO vector distribution. At the heart of this issue is the imperfectness of quasi-orthogonality. While a pair of vectors might approach orthogonality, the same cannot be confidently said for the subspaces spanned by larger sets of QO vectors.

Capitalizing on the empirical observation that real-world graphs predominantly obey the power-law distribution [9], we propose a strategy to control variance. Leveraging the prevalence of high-degree nodes—or *hubs*—we designate unique one-hot vectors for the foremost hubs. Consider the graph's top-b hubs; while other nodes draw their QO vectors from a hypercube $\{-1/\sqrt{F-b}, 1/\sqrt{F-b}\}^{F-b} \bigtimes \{0\}^b$, these hubs are assigned one-hot vectors from $\{0\}^{F-b} \bigtimes \{0,1\}^b$, reserving a distinct subspace of the linear space to safeguard orthogonality. Note that when new nodes are added, their QO vectors are sampled the same way as the non-hub nodes, which can ensure a tractable computation complexity.

Norm rescaling to facilitate weighted counts. Theorem 4 alludes to an intriguing proposition: the estimator's potential to encapsulate not just CN, but also RA. Essentially, RA and AA are nuanced heuristics translating to weighted enumerations of shared neighbors, based on their node degrees. In Theorem 5, such counts are anchored by vector norms during dot products. MPLP enhances this count methodology by rescaling node vector norms, drawing inspiration from previous works [114, 176]. This rescaling is determined by the node's representation, GNN(u), and its degree

d_u. The rescaled vector is formally expressed as:

$$\tilde{\boldsymbol{h}}_k^{(0)} = f(\mathrm{GNN}(k)||[d_k]) \cdot \boldsymbol{h}_k^{(0)}, \tag{4.4}$$

where $f\colon \mathbb{R}^{F_x+1} \to \mathbb{R}$ is an MLP mapping the node representation and degree to a scalar, enabling the flexible weighted count paradigm.

4.5.2 Structural feature estimations

Node label estimation. The estimator in Theorem 5 can effectively quantify CN. Nonetheless, solely relying on CN fails to encompass diverse topological structures embedded within the local neighborhood. To offer a richer representation, we turn to Distance Encoding (DE) [90]. DE acts as an adept labeling tool [182], demarcating nodes based on their shortest-path distances relative to a target node pair. For a given pair (u, v), a node k belongs to a node set $\mathrm{DE}(p, q)$ iff $\mathrm{SPD}(u, k) = p$ and $\mathrm{SPD}(v, k) = q$. Unlike its usage as node labels, we opt to enumerate these labels, producing a link feature defined by $\#(p, q) = |\mathrm{DE}(p, q)|$. Our model adopts a philosophy akin to ELPH [22], albeit with a distinct node-estimation mechanism.

Returning to Theorem 6, we recall that message-passing as in Equation 4.3 essentially corresponds to walks. Our ambition to enumerate nodes necessitates a single-layer message-passing alteration, reformulating Equation 4.3 to:

$$\boldsymbol{\eta}_v^{(s)} = \sum_{k \in \mathcal{N}_v^s} \tilde{\boldsymbol{h}}_k^{(0)}. \tag{4.5}$$

Here, $\mathcal{N}_v^s$ pinpoints v's shortest-path neighborhoods distanced by the shortest-path s. This method sidesteps the duplication dilemma highlighted in Theorem 6, ensuring that $\boldsymbol{\eta}_v^{(s)}$ aggregates at most one QO vector per node. Similar strategies are explored in [2, 47].

For a tractable computation, we limit the largest shortest-path distance as $r \geq$

$max(p, q)$. Consequently, to capture the varied proximities of nodes to the target pair (u, v), we can deduce:

$$\#(p,q) = \begin{cases} \mathbb{E}\big(\boldsymbol{\eta}_u^{(p)} \cdot \boldsymbol{\eta}_v^{(q)}\big), & r \geq p, q \geq 1 \\ |\mathcal{N}_v^q| - \sum\limits_{1 \leq s \leq r} \#(s,q), & p = 0 \\ |\mathcal{N}_u^p| - \sum\limits_{1 \leq s \leq r} \#(p,s), & q = 0 \end{cases} \tag{4.6}$$

Concatenating the resulting estimates yields the expressive structural features of MPLP.

Shortcut removal. The intricately designed structural features improve the expressiveness of MPLP. However, this augmented expressiveness introduces susceptibility to distribution shifts during link prediction tasks [37]. Consider a scenario wherein the neighborhood of a target node pair contains a node k. Node k resides a single hop away from one of the target nodes but requires multiple steps to connect with the other. When such a target node pair embodies a positive instance in the training data (indicative of an existing link), node k can exploit both the closer target node and the link between the target nodes as a shortcut to the farther one. This dynamic ensures that for training-set positive instances, the maximum shortest-path distance from any neighboring node to the target pair is constrained to the smaller distance increased by one. This can engender a discrepancy in distributions between training and testing phases, potentially diminishing the model's generalization capability.

To circumvent this pitfall, we adopt an approach similar to preceding works [72, 155, 171, 178]. Specifically, we exclude target links from the original graph during each training batch, as shown by the dash line in Figure 4.4. This maneuver ensures these links are not utilized as shortcuts, thereby preserving the fidelity of link feature construction.

Feature integration for link prediction. Having procured the structural features, we proceed to formulate the encompassing link representation for a target node pair (u, v) as:

$$\boldsymbol{h}_{(u,v)} = (\text{GNN}(u) \odot \text{GNN}(v))||[\#(1,1), \ldots, \#(r,r)],$$

which can be fed into a classifier for a link prediction between nodes (u, v).

4.5.3 More scalable estimation

MPLP estimates the cardinality of the distinct node sets with different distances relative to target node pairs in Equation 4.6. However, this operation requires a preprocessing step to construct the shortest-path neighborhoods $\mathcal{N}_v^s$ for $s \leq r$, which can cause computational overhead on large-scale graph benchmarks. To overcome this issue, we simplify the structural feature estimations as:

$$\#(p,q) = \mathbb{E}\Big(\tilde{\boldsymbol{h}}_u^{(p)} \cdot \tilde{\boldsymbol{h}}_v^{(q)}\Big), \tag{4.7}$$

where $\tilde{\boldsymbol{h}}_v^{(l+1)} = \sum_{u \in \mathcal{N}_v} \tilde{\boldsymbol{h}}_u^{(l)}$ follows the message-passing defined in Equation 4.3. Such an estimation only requires the one-hop neighborhood $\mathcal{N}_v$, which is provided in a format of adjacency matrices/lists by most graph datasets. Therefore, we can substitute the structural features of MPLP with the estimation in Equation 4.7. We denote such a model with walk-level features as MPLP+.

Triangular substructure estimation. Our method, primarily designed to encapsulate the local structure of a target node pair, unexpectedly exhibits the capacity for estimating the count of triangles linked to individual nodes. This capability, traditionally considered beyond the reach of GNNs, marks a significant advancement in the field [27]. Although triangle counting is less directly relevant in the context of

link prediction, the implications of this capability are noteworthy. To maintain focus, we relegate the detailed discussion on pure message-passing for effective triangle counting to Appendix C.2.

4.6 Experiments

TABLE 4.1

LINK PREDICTION RESULTS ON NON-ATTRIBUTED BENCHMARKS.

	USAir	**NS**	**PB**	**Yeast**	**C.ele**	**Power**	**Router**	**E.coli**
Metric	Hits@50	Hits@50	Hits@50	Hits@50	Hits@50	Hits@50	Hits@50	Hits@50
CN	80.52±4.07	74.00±1.98	37.22±3.52	72.60±3.85	47.67±10.87	11.57±0.55	9.38±1.05	51.74±2.70
AA	85.51±2.25	74.00±1.98	39.48±3.53	73.62±1.01	58.34±2.88	11.57±0.55	9.38±1.05	68.13±1.61
RA	85.95±1.83	74.00±1.98	38.94±3.54	73.62±1.01	61.47±4.59	11.57±0.55	9.38±1.05	74.45±0.55
GCN	73.29±4.70	78.32±2.57	37.32±4.69	73.15±2.41	40.68±5.45	15.40±2.90	24.42±4.59	61.02±11.91
SAGE	83.81±3.09	56.62±9.41	**47.26±2.53**	71.06±5.12	58.97±4.77	6.89±0.95	42.25±4.32	75.60±2.40
SEAL	**90.47±3.00**	86.59±3.03	44.47±2.86	**83.92±1.17**	**64.80±4.23**	**31.46±3.25**	**61.00±10.10**	83.42±1.01
Neo-GNN	86.07±1.96	83.54±3.92	44.04±1.89	83.14±0.73	63.22±4.32	21.98±4.62	42.81±4.13	73.76±1.94
ELPH	87.60±1.49	**88.49±2.14**	46.91±2.21	82.74±1.19	64.45±3.91	26.61±1.73	**61.07±3.06**	75.25±1.44
NCNC	86.16±1.77	83.18±3.17	46.85±3.18	82.00±0.97	60.49±5.09	23.28±1.55	52.45±8.77	**83.94±1.57**
MPLP	**92.12±2.21**	**90.02±2.04**	**52.55±2.90**	**85.36±0.72**	**74.28±2.09**	**32.66±3.58**	**64.68±3.14**	**86.11±0.83**
MPLP+	**91.24±2.11**	**88.91±2.04**	**51.81±2.39**	**84.95±0.66**	**72.73±2.99**	**31.86±2.59**	60.94±2.51	**87.07±0.89**

The format is average score ± standard deviation. The top three models are colored by **First**, **Second**, **Third**.

TABLE 4.2

LINK PREDICTION RESULTS ON ATTRIBUTED BENCHMARKS.

	CS	**Physics**	**Computers**	**Photo**	**Collab**	**PPA**	**Citation2**
Metric	Hits@50	Hits@50	Hits@50	Hits@50	Hits@50	Hits@100	MRR
CN	51.04±15.56	61.46±6.12	21.95±2.00	29.33±2.74	61.37±0.00	27.65±0.00	51.47±0.00
AA	68.26±1.28	70.98±1.96	26.96±2.08	37.35±2.65	64.35±0.00	32.45±0.00	51.89±0.00
RA	68.25±1.29	72.29±1.69	28.05±1.59	40.77±3.41	64.00±0.00	**49.33±0.00**	51.98±0.00
GCN	66.00±2.90	73.71±2.28	22.95±10.58	28.14±7.81	35.53±2.39	18.67±0.00	84.74±0.21
SAGE	57.79±18.23	74.10±2.51	33.79±3.11	46.01±1.83	36.82±7.41	16.55±0.00	82.60±0.36
SEAL	68.50±0.76	74.27±2.58	30.43±2.07	46.08±3.27	64.74±0.43	48.80±3.16	**87.67±0.32**
Neo-GNN	71.13±1.69	72.28±2.33	22.76±3.07	44.83±3.23	57.52±0.37	49.13±0.60	87.26±0.84
ELPH	72.26±2.58	65.80±2.26	29.01±2.66	43.51±2.37	65.94±0.58	OOM	OOM
NCNC	**74.65±1.23**	**75.96±1.73**	**36.48±4.16**	**47.98±2.36**	**66.61±0.71**	**61.42±0.73**	**89.12±0.40**
MPLP	**76.40±1.44**	**76.46±1.95**	**43.47±3.61**	**58.08±3.68**	**67.05±0.51**	OOM	OOM
MPLP+	**75.55±1.46**	**76.36±1.40**	**42.21±3.56**	**57.76±2.75**	**66.99±0.40**	**65.24±1.50**	**90.72±0.12**

The format is average score ± standard deviation. The top three models are colored by **First**, **Second**, **Third**.

4.6.1 Datasets, baselines and experimental setup

We conduct evaluations across a diverse spectrum of 15 graph benchmark datasets, which include 8 non-attributed and 7 attributed graphs. It also includes three datasets from OGB [62] with predefined train/test splits. In the absence of predefined splits, links are partitioned into train, validation, and test sets using a 70-10-20 percent split. Our comparison spans three categories of link prediction models: (1) heuristic-based methods encompassing CN, AA, and RA; (2) node-level models like GCN and SAGE; and (3) link-level models, including SEAL, Neo-GNN [176], ELPH [22], and NCNC [155]. Each experiment is conducted 10 times, with the average score and standard deviations reported. The evaluation metrics are aligned with the standard metrics for OGB datasets, and we utilize Hits@50 for the remaining

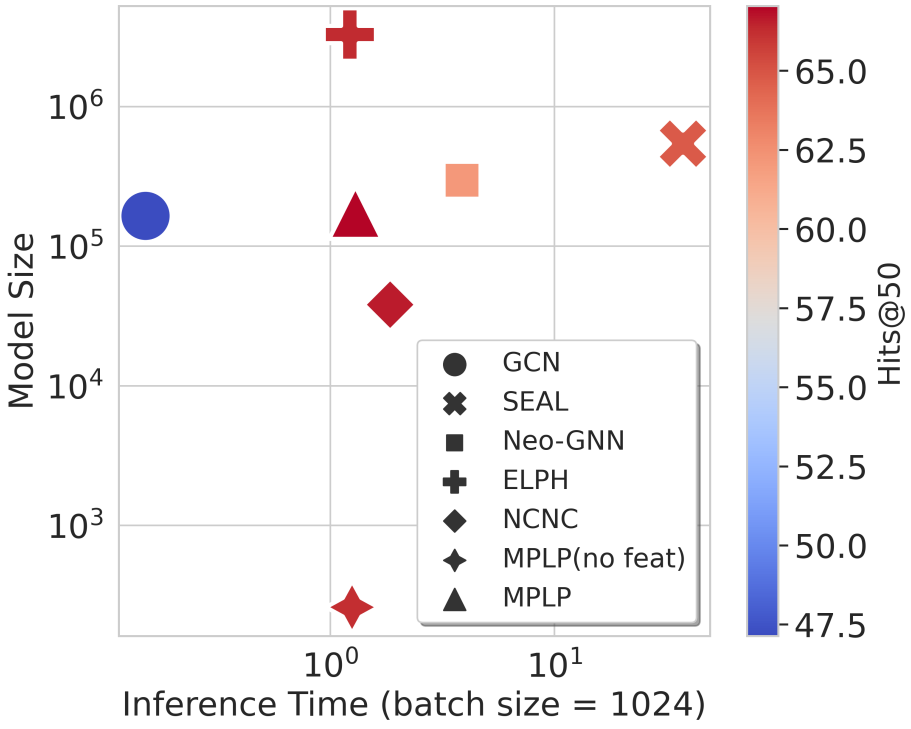

Figure 4.5: Evaluation of model size and inference time on Collab. The inference time encompasses the entire cycle within a single epoch.

datasets. We limit the number of hops $r = 2$, which results in a good balance of performance and efficiency. A comprehensive description of the experimental setup is available in Appendix C.3.

4.6.2 Results

Performance metrics are shown in Tables 4.1 and 4.2. Our methods, MPLP and MPLP+, demonstrate superior performance, surpassing baseline models across all evaluated benchmarks by a significant margin. Notably, MPLP tends to outperform MPLP+ in various benchmarks, suggesting that node-level structural features (Equation 4.6) might be more valuable for link prediction tasks than the walk-level features (Equation 4.7). In large-scale graph benchmarks such as PPA and Citation2, MPLP+ sets new benchmarks, establishing state-of-the-art results. For other

datasets, our methods show a substantial performance uplift, with improvements in Hits@50 ranging from 2% to 10% compared to the closest competitors.

4.6.3 Model size and inference time

A separate assessment focuses on the trade-off between model size and inference time using the Collab dataset, with findings presented in Figure 4.5. Observing the prominent role of graph structure in link prediction performance on Collab, we introduce a streamlined version of our model, termed MPLP(no feat). This variant solely capitalizes on structural features, resulting in a compact model with merely 260 parameters. Nevertheless, its efficacy rivals that of models which are orders of magnitude larger. Furthermore, MPLP's inference time for a single epoch ranks among the quickest in state-of-the-art approaches, underscoring its efficiency both in terms of time and memory footprint. More details can be found in Appendix C.3.3.

4.6.4 Estimation accuracy

We investigate the precision of MPLP in estimating $\#(p, q)$, which denotes the count of node labels, using the Collab dataset. The outcomes of this examination are illustrated in Figure C.4. Although ELPH possesses the capability to approximate these counts utilizing techniques like MinHash and Hyperloglog, our method exhibits superior accuracy. Moreover, ELPH runs out of memory when the dimension is larger than 3000. Remarkably, deploying a one-hot encoding strategy for the hubs further bolsters the accuracy of MPLP, concurrently diminishing the variance introduced by inherent graph structures. An exhaustive analysis, including time efficiency considerations, is provided in Appendix C.5.1.

4.6.5 Extended ablation studies

Further ablation studies have been carried out to understand the individual contributions within MPLP. These include: (1) an exploration of the distinct components of MPLP in Appendix C.5.2; (2) an analysis of the performance contributions from different structural estimations in Appendix C.5.3; and (3) an examination of parameter sensitivity in Appendix C.5.4.

4.7 Conclusion

We study the potential of message-passing GNNs to encapsulate link structural features. Based on this, we introduce a novel link prediction paradigm that consistently outperforms state-of-the-art baselines across various graph benchmarks. The inherent capability to adeptly capture structures enhances the expressivity of GNNs, all while maintaining their computational efficiency. Our findings hint at a promising avenue for elevating the expressiveness of GNNs through probabilistic approaches.

CHAPTER 5

EFFICIENT GNNS WITHOUT GRADIENT DESCENT OPTIMIZATION

5.1 Overview

This chapter addresses the efficiency of GNNs by introducing a novel training-free paradigm for semi-supervised node classification tasks on text-attributed graphs (TAG). Convention GNNs typically rely on iterative training procedures such as gradient descent, demanding substantial computational resources. This chapter investigates an alternative approach rather than iterative optimization. It directly approximates optimal GNN parameters from node attributes and graph structure. Observing that node features belonging to the same class cluster into similar linear subspaces, we propose a closed-form solution that directly derives optimal model parameters from these observed subspace properties. This eliminates the computationally expensive training process, substantially improving the practicality and scalability of GNN models. The empirical results show that our training-free approach achieves competitive predictive performance while significantly reducing computational overhead.

This chapter primarily builds upon a previously published work [40] in collaboration with Zhichun Guo and Nitesh V. Chawla.

5.2 Introduction

Graph structured data is widely used across many fields due to its ability to show relationships between different entities [165]. In many cases, the nodes in these graphs

are associated with text attributes, leading to what's known as text-attributed graphs (TAG) [28]. TAG has versatile applications, including social media [91], citation networks [169], academic collaborations [62], or recommendation system [131].

We aim to delve into effective approaches for handling TAG, focusing specifically on semi-supervised node classification tasks [169]. The objective here is to predict the labels of unlabeled nodes within a graph, utilizing a limited set of labeled nodes as a reference [80]. To effectively capture both the node attributes and the topological structure within the graph, Graph Neural Networks (GNNs) [59, 82, 149], especially those of the message-passing type [57], have demonstrated significant success in effectively managing graph-structured data. Typically, the GNN process begins by transforming the textual data of each node into a vector using text embedding techniques like Bag-Of-Words (BOW), TF-IDF, and word2vec [104][1]. The straightforward application of these shallow text embedding methods has led to their widespread adoption as the go-to text encoding technique in numerous graph benchmark datasets [62, 131, 169] (see Table E.1).

When it comes to node classification with the textual data encoded, a GNN is trained on the graph to fit the data accurately. The training phase, often seen as an optimization process, commonly employs iterative tools like gradient descent to update the model's weight to minimize a predefined loss function over the training samples. Although gradient descent (and its variants [139]) has become almost synonymous with model fitting, it raises a question whether there are alternative methods to fit the GNNs.

One such alternative is proposed by UGT [66], drawing inspiration from the lottery ticket hypothesis [52, 190]. UGT suggests fitting GNNs without updating model weights by identifying a mask to sparsify untrained neural networks. Although these

[1]Recently, there has been growing interest in utilizing Large Language Models to encode textual data. However, employing a more complex text encoder alone doesn't offer substantial benefits over simpler embeddings like BOW and TF-IDF [28, 122]

sparse subnetworks can perform comparably to trained dense networks, finding an appropriate mask involves an iterative discrete optimization process, which can be even more computationally demanding than traditional gradient descent optimization.

In this paper, we explore how **GNNs can be fitted without employing traditional iterative processes like gradient descent to tackle semi-supervised node classification tasks on TAG**. Given the absence of a closed-form solution for multiclass classification problems, we suggest approximating optimal parameters by harnessing both the node attributes and graph structures. We observe that on TAG, text encodings from the same class tend to cluster in the same linear subspace, while being orthogonal to those from different classes. Moreover, we investigate the training dynamics of GCN [82] and SGC [161], two representative GNNs. Our findings suggest that traditional GNN training on TAG can be seen as a process to locate the weight vectors close to the text encodings from corresponding classes in the linear space. Inspired by these insights, we introduce TrainlessGNN, a method that fits a linear GNN model by constructing a weight matrix reflecting the subspace of a particular class's node attributes. The formulation of the weight matrix in our approach can be interpreted as a closed-form solution for a linear regression problem, solved by minimum-norm interpolation in an over-parameterized regime [151], offering a novel pathway for addressing semi-supervised node classification on TAG.

To summarize, our contributions are as follows:

- We investigate the training dynamics of common GNNs like GCN and SGC on TAG. We discover that the weight matrix is fundamentally pushed towards approximating the subspaces of the node attributes associated with respective classes.
- We introduce TrainlessGNN, an innovative and efficient method for semi-supervised node classification. To our knowledge, TrainlessGNN is the first to achieve significant predictive performance without the need for an iterative training process in fitting GNN models.
- Through empirical evaluation on various TAG benchmarks, we demonstrate that our method, devoid of a typical training process, can either match or

surpass the performance of conventionally trained models.

5.3 Preliminaries and Related Work

Notations. We examine a graph $\mathcal{G} = (\mathcal{V}, \mathbf{A}, \mathbf{X})$. The graph is comprised of a node set $\mathcal{V}$ with a cardinality of n, indexed as $\{v_1, \ldots, v_n\}$. The adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ characterizes the structural relationships between nodes. We further define the degree matrix $\mathbf{D}$ as a diagonal matrix, with $\mathbf{D} = \text{diag}(d_1, \ldots, d_n)$. Every node v_i is associated with a d-dimensional feature vector $\mathbf{x}_i \in \mathbb{R}^d$. When combined, these vectors form the feature matrix as $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^\top \in \mathbb{R}^{n \times d}$. The label of each node y_i belongs to one among the C classes, enumerated as $\{1, 2, \ldots, C\}$. We denote the one-hot encoding matrix of the labels as $\mathbf{B}$.

Semi-supervised node classification. In the context of our work, we tackle a semi-supervised node classification task. The entire node set $\mathcal{V}$ is divisibly partitioned into two discrete subsets: $\mathcal{U}$, representing the unlabeled nodes, and $\mathcal{L}$, encompassing the labeled nodes. Similarly, the original feature matrix $\mathbf{X}$ is divided into $\mathbf{X}_\mathcal{L}$ and $\mathbf{X}_\mathcal{U}$, corresponding to the node sets they belong to. Our primary goal is to leverage the labeled subset $\mathcal{L}$ to predict class labels for the nodes in $\mathcal{U}$ with unknown labels. Beyond traditional classification tasks, the semi-supervised node classification task is confronted with a heightened challenge that there exists a predominant presence of unknown labels within the testing set compared to the limited known labels within the training set. This configuration echoes the settings explored in prior studies [62, 131, 169].

Graph Neural Networks. Graph neural networks (GNNs) are a family of algorithms that extract structural information from graphs that encode graph-structured data into node representations or graph representations. They initialize each node fea-

ture representation with its attributes $\mathbf{H}^{(0)} = \mathbf{X}$ and then gradually update it by aggregating representations from its neighbors. Formally, given a graph $\mathcal{G} = (\mathcal{V}, \mathbf{A}, \mathbf{X})$, the l-th layer GNN is defined as:

$$\mathbf{H}^{(l)} \coloneqq \mathrm{UPD}(\mathbf{H}^{(l-1)}, \mathrm{AGG}(\mathbf{H}^{(l-1)}, \mathbf{A}))), \tag{5.1}$$

where $\mathrm{AGG}(\cdot)$ and $\mathrm{UPD}(\cdot)$ denote the neighborhood aggregation function and the updating function respectively [57]. As a result, the final layer output of a GNN, represented as $\mathbf{Z} = \mathbf{H}^{(l)} \in \mathbb{R}^{n \times C}$, serves as the predicted logits for different classes. To derive a prediction, one can select the class label associated with the highest logit for each node:

$$\hat{y}_i = \arg\max_c \mathbf{Z}_{i,c}. \tag{5.2}$$

Decoupled GNNs. Prominent GNNs such as GCN typically incorporate learnable MLPs within the $\mathrm{UPD}(\cdot)$ function across each layer. Nonetheless, recent studies [188] suggest that decoupling message passing from feature transformation can deliver competitive performance when benchmarked against traditional GNNs. These so-called Decoupled GNNs (DeGNNs) are often characterized by their computational efficiency and are better to scale with larger graphs. Broadly, DeGNNs fall into two categories, distinguished by the sequence in which they conduct message passing and feature transformation.

For example, the SGC model [161] first undertakes multiple rounds of message passing before culminating with a trainable linear layer for prediction. It performs the message-passing step similar to GCN but without layer-wise linear transformation as:

$$\mathbf{H}^{(l)} = \left(\tilde{\mathbf{A}}\right)^L \mathbf{X}, \mathbf{Z} = \mathbf{H}^{(l)}\mathbf{W}, \tag{5.3}$$

where $\tilde{\mathbf{A}} = \hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}$ and $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the adjacency matrix with added self-loops. $\mathbf{W} \in \mathbb{R}^{d \times C}$ is the learnable weight matrix.

Conversely, the C&S approach [64] initiates training with an MLP exclusively on node attributes, devoid of the graph structure. It then propagates the resulting logits through the graph, refining prediction based on the graph structure as:

$$\hat{\mathbf{Z}} = \text{MLP}(\mathbf{X}), \mathbf{Z} = \text{C\&S}(\hat{\mathbf{Z}}, \mathbf{A}), \tag{5.4}$$

where $\hat{\mathbf{Z}}$ is the logits for the node classification tasks. For those keen on the specific formulation of C&S($\mathbf{Z_0}, \mathbf{A}$), we have detailed it in the appendix.

Objective and Training Process. To train GNNs, one must optimize their weight matrices to fit on the dataset. Surrogate measures such as cross entropy, depicted in Equation 5.5, are employed to iteratively adjust the model weights to reduce discrepancies between its predictions and the true labels on the labeled subset $\mathcal{L}$ of the graph.

$$\text{Loss} = -\frac{1}{|\mathcal{L}|} \sum_{v_i \in \mathcal{L}} \log \left(\frac{e^{\mathbf{Z}_{i,y_i}}}{\sum_{j=1}^{C} e^{\mathbf{Z}_{i,j}}} \right). \tag{5.5}$$

5.4 Unpacking What GNNs Learn on Text-Attributed Graphs

In this section, our goal is to gain insights into the inner workings of GNNs, especially during training for node classification on graphs that have text attributes. We will first discuss common text encoding methods used to convert textual information into a format suitable for machine learning. Then, we will examine the behavior of weight matrices in two GNN models: SGC and GCN. Insights from this exploration will pave the way for our proposed method, which does not require to optimize the loss function to fit the data.

5.4.1 Quasi-orthogonal node attributes

On a TAG, nodes contain text descriptions that highlight their unique characteristics. But to make this text data useful for machine learning algorithms, we convert it into vectors. Node attributes in such a graph are commonly encoded using methods like Bag-of-Words (BOW) and TF-IDF [62, 131, 169]. The essence of these methods is to construct a vocabulary from word tokens and then encode documents based on word token occurrences. For expansive vocabularies, such encoding tends to be sparse. This sparsity implies that two documents with differing lexicons are likely to yield encodings that are orthogonal. We term this attribute behavior as *quasi-orthogonality* (QO). We sought to empirically ascertain the QO

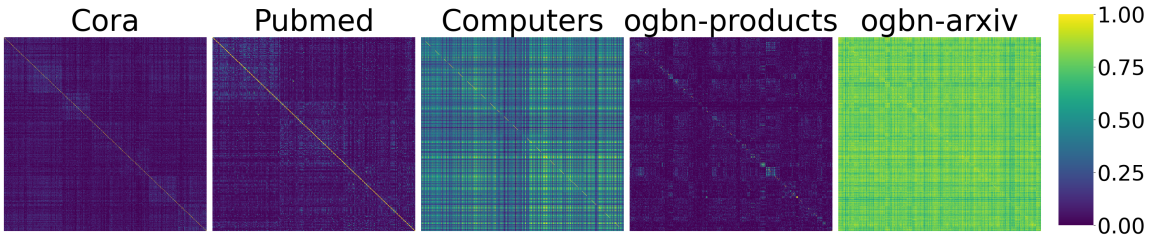

Figure 5.1: Heatmap of the inner product of node attributes on TAG.

of node attributes across various TAG. Our analysis spanned five distinct datasets: Cora (BOW encoded), Pubmed (TF-IDF encoded) [169], Computers (BOW encoded but with relatively smaller vocabulary) [131], OGBN-Products (BOW encoded followed by principle component analysis), and OGBN-Arxiv (encoded via word2vec averaging) [62].

The heatmap in Figure 5.1 plots the inner products of node attributes. Both the x and y axes denote node indices, whereas the color gradient signifies the magnitude

of their inner product. We further order nodes by their true labels, y, ensuring nodes from identical classes cluster adjacently.

In the datasets of Cora, Pubmed, and OGN-Product, the heatmap clearly exhibits QO. The inner product between attributes of any node pair is almost negligible. Moreover, brighter blocks along the diagonal indicate that nodes within the same classes have a tendency to exhibit higher inner products compared to other nodes. In contrast, the OGBN-Arxiv dataset doesn't demonstrate this QO trait. This deviation can be attributed to the use of average word embeddings for node attribute generation, which can compromise the QO among nodes. Interestingly, despite being encoded with BOW, the Computers dataset lacks prominent QO node attributes. This might stem from its relatively smaller vocabulary size. In the experiment section, we will illustrate that the QO property plays a critical role in determining the efficacy of our proposed trainless methods compared to the trained approaches.

5.4.2 What SGC learns

In this section, we probe deeper into the learning dynamics of the SGC model when trained via gradient descent. Notably, the SGC model comprises a sole trainable weight matrix, $\mathbf{W} = (\mathbf{W}_{:,1}, \ldots, \mathbf{W}_{:,C}) \in \mathbb{R}^{d \times C}$, as illustrated in Equation 5.3. Our primary focus lies in understanding the interplay between the node attributes from the training set and this weight matrix.

The logit $\mathbf{Z}_{i,c}$ for node i concerning class c is computed by the inner product $\mathbf{Z}_{i,c} = \mathbf{x}_i \cdot \mathbf{W}_{:,c}$. For a prediction to be accurate, we would anticipate that the inner product with the weight vector corresponding to the true class surpasses those of other classes. Given the QO observed in node attributes, this phenomenon might be even more pronounced. We base our experiment on the Cora dataset, characterized by 7 label classes with 20 labeled nodes for each class [169]. We evaluate the inner product of each column vector $\mathbf{W}_{:,c}$ of the weight matrix (for $1 \leq c \leq 7$) against the node

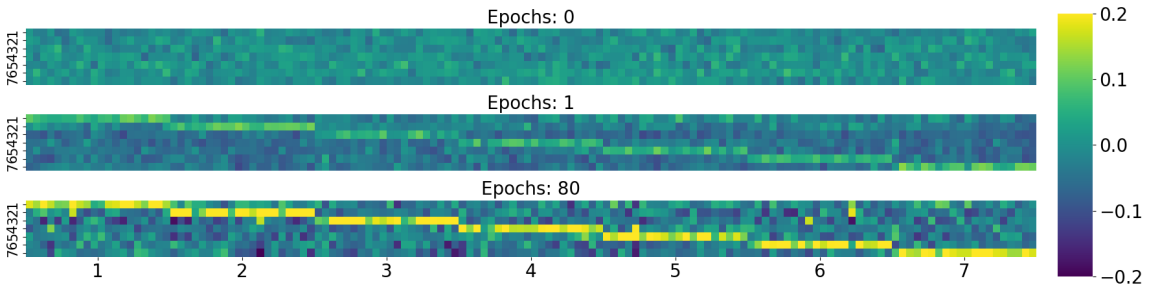

Figure 5.2: Heatmap depicting the evolution of inner products between node attributes and the weight vectors across various training epochs on the Cora dataset.

attributes $\mathbf{x}_i$ for nodes $v_i \in \mathcal{L}$. This results in a heatmap of 7 rows and $20 \times 7 = 140$ columns, with nodes of identical labels grouped together. The progression of this heatmap across various epochs during training is displayed in Figure 5.2.

The evolving heatmaps reveal a pattern: as training progresses, the weight matrix inclines to heighten the inner product between a node v_i's attribute and its corresponding class's weight vector, $\mathbf{W}_{:,y_i}$, while diminishing the product with other classes. This amplifies the logit for the true class y_i, suppressing logits for other classes towards zero, subsequently reducing the loss in Equation 5.5. This observation serves as a foundation for our subsequent proposal, where we seek to derive the weight matrix directly from the aggregation of node attributes.

5.4.3 What GCN learns

We previously observed the SGC's propensity to optimize its weight matrix, ensuring a heightened inner product between node attributes and the corresponding class's weight vector. This section delves into discerning whether GNNs, particularly those with non-linearities and multiple layers, exhibit analogous learning dynamics. We direct our focus to the popularly employed 2-layer GCN.

TABLE 5.1

COMPARISON OF ACCURACY.

Dataset	Cora	Citeeer	Pubmed
SGC	81.00	71.90	78.90
2-layer GCN	81.50	71.40	78.50
second-layer-frozen GCN	80.40	70.50	77.80

Comparison of accuracy between SGC, 2-layer GCN, and 2-layer GCN with the second layer frozen.

We first rewrite the Equation 5.1 for GCN as:

$$\mathbf{H}^{(1)} = \sigma(\tilde{\mathbf{A}}\mathbf{X}\mathbf{W}^{(1)}), \quad \mathbf{H}^{(2)} = \tilde{\mathbf{A}}\mathbf{H}^{(1)}\mathbf{W}^{(2)}. \tag{5.6}$$

Interestingly, the one-layer model, such as the SGC, achieves comparable performance with the 2-layer GCN, as shown in Table 5.1. This leads us to question the necessity of simultaneously training weight matrices for both layers. To explore this, we train a 2-layer GCN, but keep the second layer's weight matrix frozen, preserving its initial state. The outcomes, shown in Table 5.1, reveal that performance remains robust even with a static second layer. This suggests that in scenarios where node attributes are sufficiently learned by a linear model like SGC, the MLP integrated within GCNs might be redundant.

Pursuing this inquiry, with the second layer's weight matrix still frozen, we undertake an experiment analogous to our earlier one on SGC. Unlike our previous focus on the inner product between node attributes $\mathbf{X}$ and weight vectors $\mathbf{W}_{:,c}$, we now assess the correlation between the node representation $\mathbf{H}^{(1)}$ from the GCN's first layer and the second layer's weight matrix $\mathbf{W}^{(2)}$. Specifically, we calculate the inner product

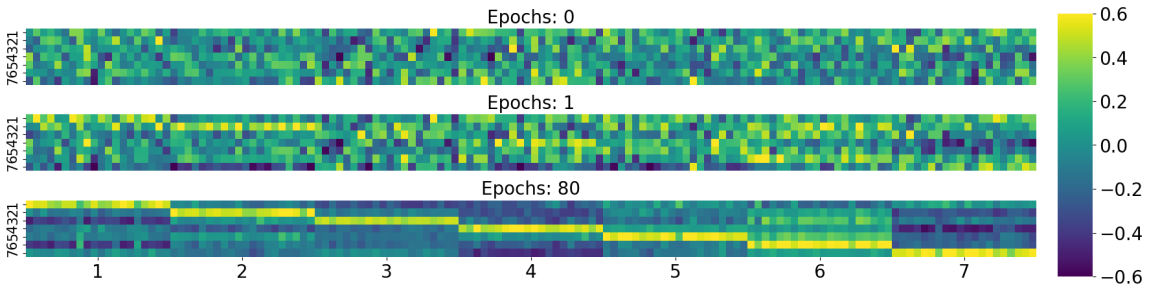

Figure 5.3: Heatmap depicting the evolution of inner products between node representations from GCN's first layer and the second layer's weight vectors across various training epochs on the Cora dataset.

between each training node's representation $\mathbf{H}^{(1)}_{i,:}$ and the column vector $\mathbf{W}^{(2)}_{:,c}$ of the weight matrix, then similarly represent this using a heatmap in Figure 5.3.

Interestingly, the first layer of the GCN appears to forge a relationship with the second layer's weight matrix that mirrors SGC's dynamics. Notably, the inner product between the node representation $\mathbf{H}^{(1)}_{i,:}$ and its associated class's weight vector $\mathbf{W}^{(2)}_{:,y_i}$ is markedly higher than with the weight vectors of other classes. Throughout the training phase, the GCN's first layer essentially learns to project node attributes from different classes into different subspaces. These subspaces are inherently defined by the corresponding randomly initialized weight vector $\mathbf{W}^{(2)}_{:,c}$ of the second layer. Given that randomly initialized vectors tend to be QO with high probability [38], this fosters a high inner product between $\mathbf{H}^{(1)}_{i,:}$ and $\mathbf{W}^{(2)}_{:,y_i}$, but nearly nullifies the product with other classes. This matches with our SGC observations.

5.5 Method

Previous analyses of SGC and GCN highlight that in cases where node attributes from different classes are nearly orthogonal, gradient descent training tends to align the weight vectors with corresponding class node attributes. It's also observed that for GCNs, freezing the last layer suggests that a sole linear layer can suffice for

TAG. Based on these insights, we introduce a simple yet efficient method, referred to as TrainlessGNN. This method creates the linear weight matrix directly from node attributes, eliminating the need for the usual gradient descent training in GNNs, while still being suitable for inference. It's applicable to linear classification models and any De-GNN with a linear layer. Further details about the implementation can be found in Appendix.

5.5.1 Building the Weight Matrix

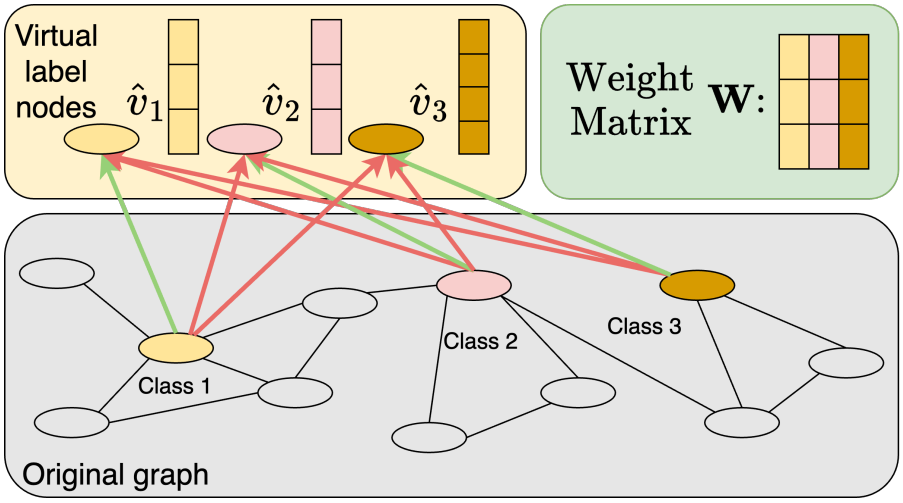

Figure 5.4: This figure outlines the process for obtaining the weight matrix $\mathbf{W}$ in TrainlessGNN. Initially, virtual label nodes are added for each class label. These nodes are then connected to labeled nodes sharing the same class, depicted by green lines. Additionally, virtual label nodes are connected to all other labeled nodes, represented by red lines, with an assigned edge weight ω. A single round of message passing updates the representation of the virtual label nodes, providing the desired weight matrix $\mathbf{W}$.

Virtual label nodes. To construct a weight matrix $\mathbf{W} \in \mathbb{R}^{d \times C}$ applicable to any linear model, it's essential to formulate the weight vectors $\mathbf{W}_{:,c} \in \mathbb{R}^d$ for $1 \leq c \leq C$. It begins by adding C *virtual label nodes* into the original graph as $\mathcal{V}_{new} = \mathcal{V} \cup \{\hat{v}_c | 1 \leq c \leq C\}$. Every virtual label node $\hat{v}_c$ symbolizes the corresponding class c, initialized as a zero vector $\mathbf{0} \in \mathbb{R}^d$. Subsequent to this, each virtual label node is connected to labeled nodes $\mathcal{L}$ possessing the matching labels. For example, the virtual label node $\hat{v}_c$ connects with all nodes from the training set labeled as c, $\{v_i \in \mathcal{L} | y_i = c\}$. The green lines in Figure 5.4 depict such connections.

Message passing. In this newly formed graph with virtual label nodes, the weight matrix is constructed by executing a single round of message passing, which essentially updates the representation of the virtual label nodes. The updated virtual label nodes representation then defines $\mathbf{W}$:

$$\mathbf{W}^\top = \mathbf{B}_{\mathcal{L}}^\top \mathbf{X}_{\mathcal{L}}, \tag{5.7}$$

where $\mathbf{X}_{\mathcal{L}}$ represents the node attributes from the labeled node sets $\mathcal{L}$. The $\mathbf{B}_{\mathcal{L}} \in \mathbb{R}^{|\mathcal{L}| \times C}$ is a one-hot encoding matrix of the labels of $\mathcal{L}$, acting as the incidence matrix between virtual label nodes $\{\hat{v}_c | 1 \leq c \leq C\}$ and labeled nodes $\mathcal{L}$. Through this approach, the weight vectors are essentially constructed based on the node attributes from the corresponding classes, with the aim to maximize the inner product between the node attribute and the weight vector of the same class.

Connecting nodes with different labels. Adjusting the weight vectors based on the node attributes from the same classes maximizes the inner product but fails to minimize the inner product for nodes from different classes. To address this, we extend the connections of virtual label nodes to other labeled nodes in the training set (red lines in Figure 5.4). Contrarily to the connections within the same class

nodes, we assign an edge weight $\omega \in \mathbb{R}$ as a hyperparameter to the newly formed connections between virtual label nodes and differently labeled nodes. More formally, the weight matrix is computed as:

$$\mathbf{W}^\top = (\mathbf{B}_\mathcal{L} - \frac{\omega}{C}\mathbf{1})^\top \mathbf{X}_\mathcal{L}. \tag{5.8}$$

In this equation, each entry of the one-hot encoding matrix $\mathbf{B}_\mathcal{L}$ is subtracted by $\frac{\omega}{C}$. By setting ω to a negative value, we can achieve a weight matrix that not only maximizes the inner product of the weight vectors and node attributes from the same class but also minimizes the inner product from different classes.

Inference. After obtaining the trainless weight matrix $\mathbf{W}$, we proceed to an accurate and efficient computation of the logits for the unlabeled node sets $\mathcal{U}$. The logits are calculated with various backbone models as detailed below:

- **Trainless Linear:** The logits $\mathbf{Z}$ are directly computed using the expression:

  $$\mathbf{Z} = \mathbf{X}\mathbf{W}, \tag{5.9}$$

  where $\mathbf{X}$ is the original feature matrix.

- **Trainless SGC:** The logits $\mathbf{Z}$ are inferred using the updated node representations $\mathbf{H}^{(l)}$ with:

  $$\mathbf{Z} = \mathbf{H}^{(l)}\mathbf{W}, \tag{5.10}$$

- **Trainless C&S:** The logits $\mathbf{Z}$ are obtained using the C&S function applied to the product of the original feature matrix $\mathbf{X}$ and the weight matrix $\mathbf{W}$ alongside the adjacency matrix $\mathbf{A}$:

  $$\mathbf{Z} = \text{C\&S}(\mathbf{X}\mathbf{W}, \mathbf{A}). \tag{5.11}$$

5.5.2 A View from Linear Regressions

In this section, we explore the equivalence between our approach and a linear classifier trained through gradient descent with a cross-entropy loss. We provide a rationale for our method, viewing it through the lens of linear regression. The dis-

cussion begins by outlining the following assumptions related to the semi-supervised node classification task:

Assumption 1. *Consider a graph* $\mathcal{G} = (\mathcal{V}, \mathbf{A}, \mathbf{X})$*, where* $\mathbf{X} \in \mathbb{R}^{n \times d}$*. We propose that:*

1. *The model is over-parametrized as the number of features* d *is adequately large, i.e.,* $d > |\mathcal{L}|$*, where* $\mathcal{L}$ *is the training/labeled set.*
2. *The row vectors of the feature matrix are orthogonal as* $\mathbf{X}_{\mathcal{L}}\mathbf{X}_{\mathcal{L}}^{\top} = \mathbf{I}$*.*

These assumptions are modest and align with most real-world scenarios. The first assumption pertinently applies to the majority of TAG created using shallow text encoding methods like BOW or TF-IDF. Here, the number of features is contingent on the size of the vocabulary. Additionally, in many semi-supervised node classification task setups, the number of nodes in the training set is often comparatively low [62, 131, 169], further accentuating the model's over-parameterization (refer to Table E.1). The second assumption, grounded in previous observations, suggests that node attributes are likely orthogonal to each other (see Figure 5.1).

In a simplified form, the classification problem in Equation 5.5 can be naively converted into a linear regression problem with the least squared loss:

$$\hat{\mathbf{W}} = \arg\min_{\mathbf{W}} \|\mathbf{X}_{\mathcal{L}}\mathbf{W} - \mathbf{B}_{\mathcal{L}}\|_2.$$

Thanks to the over-parameterization, the training loss can be reduced to zero by $\mathbf{X}_{\mathcal{L}}\hat{\mathbf{W}} = \mathbf{B}_{\mathcal{L}}$. Within this framework, our method is viewed as an effort to transpose the $\mathbf{X}_{\mathcal{L}}$ term to the right-hand side. However, while our assumption holds that $\mathbf{X}_{\mathcal{L}}\mathbf{X}_{\mathcal{L}}^{\top} = \mathbf{I}$, it does not assert that $\mathbf{X}_{\mathcal{L}}^{\top}\mathbf{X}_{\mathcal{L}} = \mathbf{I}$. As a result, our solution cannot be straightforwardly derived from this linear regression format.

To robustly affirm the effectiveness of our methodology, we employ the minimum-norm interpolation method. Given the aforementioned assumptions, we can reformu-

late the weight matrix $\mathbf{W}$ derived by our method in Equation 5.7 as:

$$\mathbf{W} = \mathbf{X}_{\mathcal{L}}^{\top}\mathbf{B}_{\mathcal{L}} = \mathbf{X}_{\mathcal{L}}^{\top}\mathbf{I}\mathbf{B}_{\mathcal{L}} = \mathbf{X}_{\mathcal{L}}^{\top}(\mathbf{X}_{\mathcal{L}}\mathbf{X}_{\mathcal{L}}^{\top})^{-1}\mathbf{B}_{\mathcal{L}}. \tag{5.12}$$

Essentially, the formulation above acts as an estimator for the linear regression task, addressed by the minimum-norm interpolation method [151]:

$$\hat{\mathbf{W}} = \arg\min_{\mathbf{W}} \|\mathbf{W}\|_2 \ , \ \text{s.t. } \mathbf{X}_{\mathcal{L}}\mathbf{W} = \mathbf{B}_{\mathcal{L}}. \tag{5.13}$$

Moreover, the weight matrix $\mathbf{W}$ acquired through the minimum-norm interpolation and in Equation 5.7 is equivalent to that achieved by other standard training methods, including SVMs or gradient descent with diverse losses such as cross-entropy, with sufficient over-parameterization [151]. This equivalency bolsters the legitimacy of our methodology.

In summary, our technique essentially transforms a convex optimization problem lacking closed-form solutions (Equation 5.5) into a linear regression (Equation 5.13) with a closed-form solution (Equation 5.7). This transformation is grounded on the distinctive sparse encodings of the TAG data.

5.6 Experiments

In this section, we evaluate TrainlessGNN on different benchmark datasets. We start by evaluating our method on nine TAG datasets with different scales.

5.6.1 Experimental setups

Datasets. We select nine commonly used TAGs as our benchmarks. We use the citation network Planetoid datasets [169], including Cora, Citeseer and Pubmed. We also use the datasets introduced by [131], including two Coauthor datasets, CS and

TABLE 5.2

RESULTS OF SEMI-SUPERVISED NODE CLASSIFICATION ON BENCHMARK DATASETS.

	Cora	Citeseer	Pubmed	CS	Physics	Computers	Photo	OGBN-Products	OGBN-Arxiv
LP	68.00±0.00	45.30±0.00	63.00±0.00	73.60±3.90	86.60±2.00	70.80±8.10	72.60±11.10	74.34±0.00	68.32±0.00
GCN	80.94±0.45	69.24±0.74	76.62±0.30	90.01±1.08	92.10±1.42	**82.46±1.66**	**88.10±1.48**	75.64±0.21	**71.74±0.29**
SAGE	80.03±0.70	69.27±0.99	76.59±0.32	89.76±0.61	91.18±1.52	81.19±2.03	87.58±2.21	**78.29±0.16**	**71.49±0.27**
Linear	59.20±0.20	60.70±0.10	72.70±0.16	87.64±0.68	87.83±1.16	57.55±5.23	76.51±2.59	46.45±0.52	41.78±0.23
SGC	**81.00±0.00**	71.90±0.10	78.90±0.00	90.60±0.96	92.66±0.89	**82.33±1.39**	**89.64±2.05**	70.67±0.20	67.63±0.32
C&S	78.40±0.00	69.70±0.00	75.40±0.00	**91.32±1.29**	92.13±2.57	70.70±11.01	85.09±4.02	**82.54±0.03**	71.26±0.01
Trainless Linear	59.10±0.00	63.10±0.00	72.40±0.00	87.97±0.66	88.06±1.05	62.12±1.84	73.38±2.60	37.12±0.00	41.57±0.00
Trainless SGC	79.60±0.00	**73.00±0.00**	76.40±0.00	**91.22±0.56**	92.74±1.37	77.32±1.73	83.45±1.73	60.48±0.00	61.71±0.00
Trainless C&S	77.90±0.00	68.40±0.00	75.30±0.00	88.89±0.54	**93.12±0.76**	78.91±1.41	87.06±2.32	77.27±0.00	69.52±0.00
Use both training and validation labels									
Trainless Linear	68.20±0.00	71.20±0.00	**79.20±0.00**	88.99±0.59	89.33±0.47	68.28±1.21	76.17±1.80	37.57±0.00	42.84±0.00
Trainless SGC	**82.70±0.00**	**77.20±0.00**	**81.30±0.00**	**91.38±0.67**	**92.93±0.61**	79.21±0.50	84.75±2.54	60.51±0.00	62.56±0.00
Trainless C&S	**83.80±0.00**	**73.20±0.00**	**79.90±0.00**	88.16±0.40	**93.49±0.20**	**81.67±0.96**	**88.69±0.78**	**77.90±0.00**	**71.70±0.00**

Results are evaluated by accuracy. The format is average score ± standard deviation. The top three models are colored by **First**, **Second**, **Third**.

Physics, and the Amazon co-purchase networks, Computers and Photo. We further include two OGB datasets [62] such as OGBN-Products and OGBN-Arxiv.

Baseline models. We select LP [194] as the graph-Laplacian baseline. We choose GCN [82] and SAGE [59], two of the most representative GNNs, as the non-linear baseline models. We then select logistic regression (denoted as **Linear**), **SGC**, and **C&S** as the linear baseline models. We implement three types of TrainlessGNN, including **Trainless Linear**, **Trainless SGC**, and **Trainless C&S**, corresponding to the trainless versions of the linear baseline models.

Evaluation protocols. We evaluate the models based on the accuracy of the test set. For datasets with predefined train/test splits (Planetoid and OGBN datasets), we follow their splits and run the evaluation 10 times for different model initializa-

tions. For datasets without predefined splits, we follow previous studies of semi-supervised node classification tasks [131], splitting the labeled nodes into training/validations/testing sets for 10 splits. For each split, we randomly pick 20/30 nodes from each class label as the training/validation sets and leave the rest as the testing sets. We then evaluate the model performance on 10 splits and report the average and standard deviations of the accuracy.

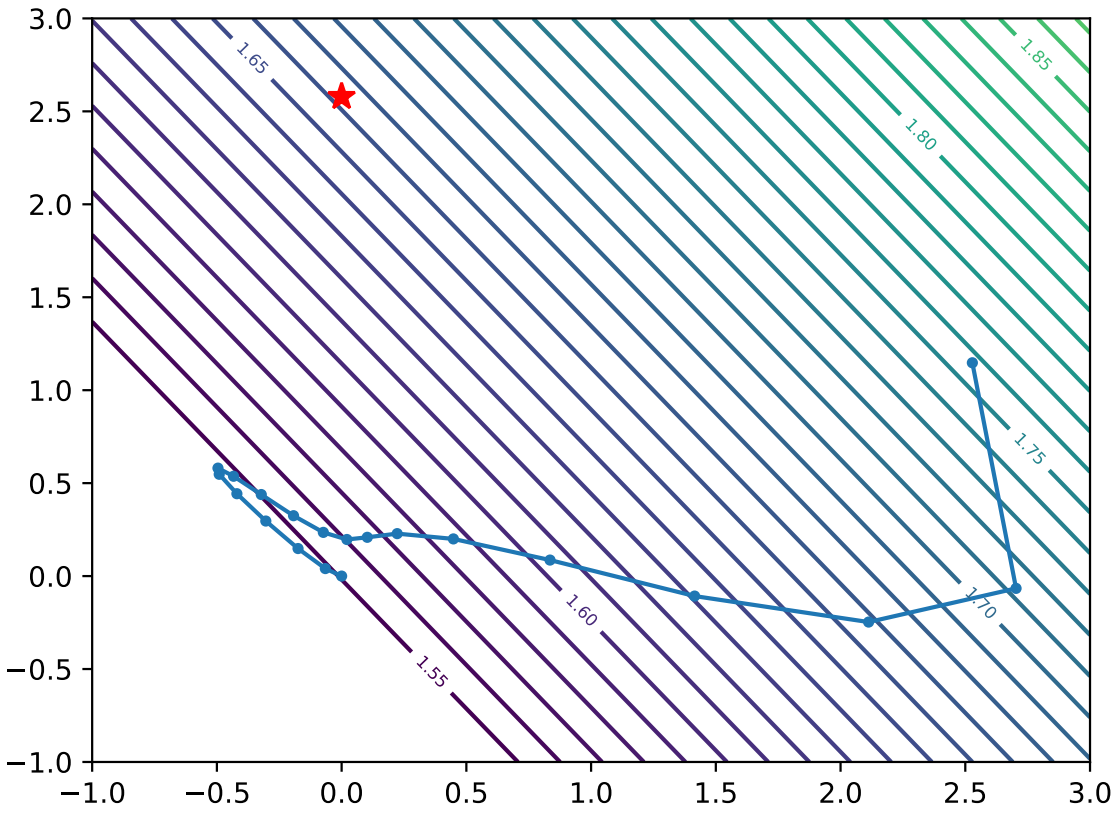

Figure 5.5: Training loss landscape while training SGC on Citeseer. The red star ($\star$) denotes Trainless SGC.

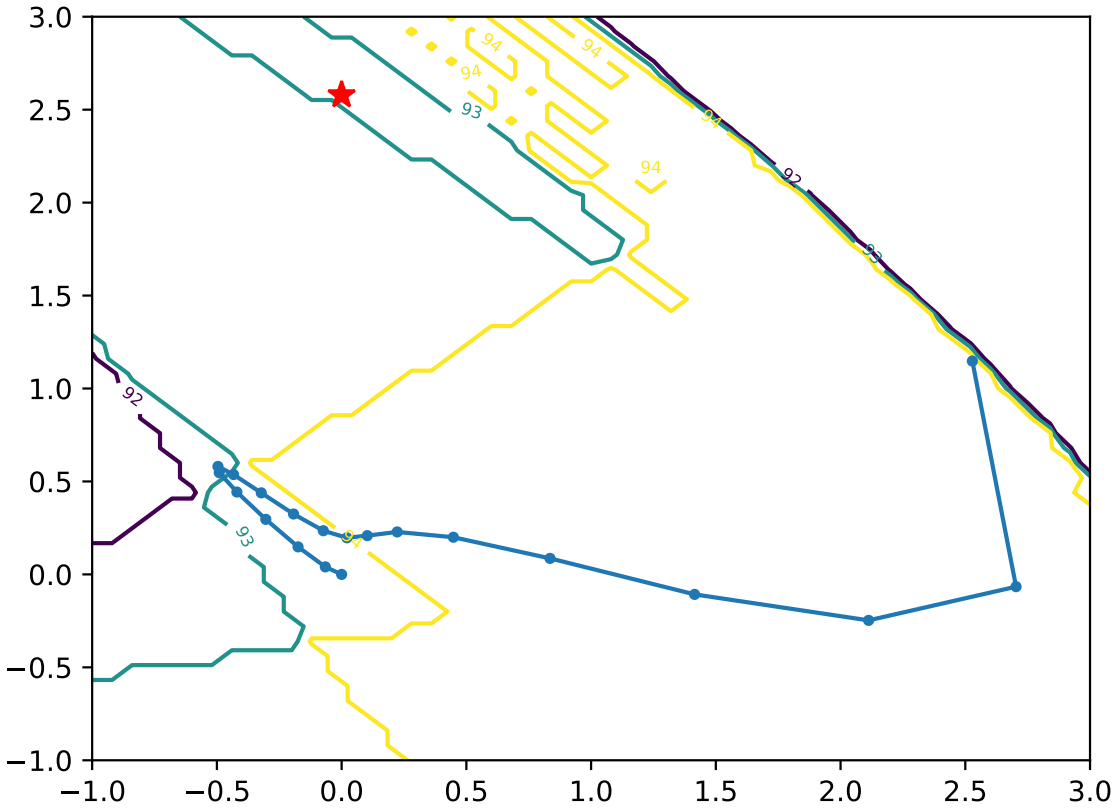

Figure 5.6: Training accuracy landscape while training SGC on Citeseer. The red star ($\star$) denotes Trainless SGC.

5.6.2 Results

We present our results in two parts. Initially, we fit TrainlessGNN using only the labeled nodes from the training set, which is the conventional approach for training GNNs. This is favorable to baseline models. Subsequently, we include labeled nodes from both training and validation sets to fit the model. This comparison remains fair as TrainlessGNN, with only tunable hyperparameter ω, is less likely to overfit on training data, eliminating the need for an exclusive validation set to prevent overfitting. Conversely, typical neural networks, especially in over parameterized domains, are prone to overfitting to zero loss, necessitating a validation set for model generalization. The results of both scenarios are shown in Table 5.2.

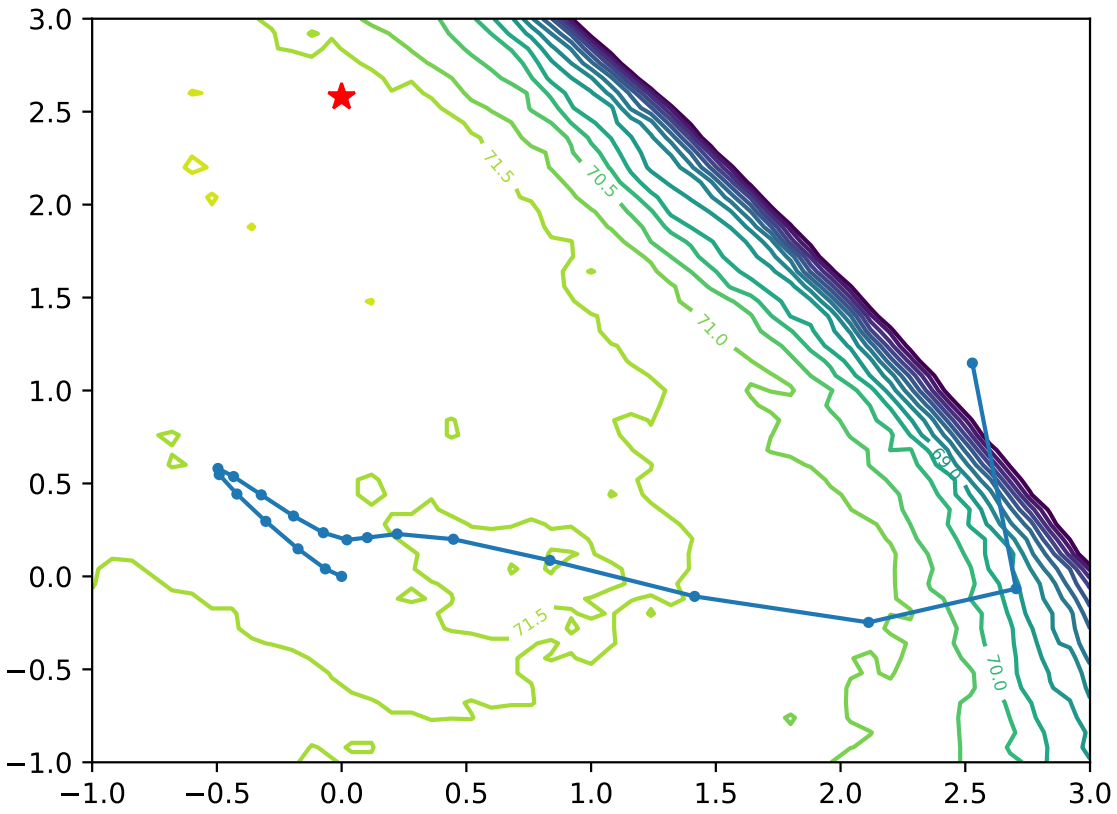

Figure 5.7: Testing accuracy landscape while training SGC on Citeseer. The red star ($\star$) denotes Trainless SGC.

TrainlessGNN on training sets. When fit on the training set, TrainlessGNN achieves comparable performance across various benchmarks to trained models. Specifically, on Cora and Pubmed, TrainlessGNN matches the performance of trained models, and notably surpasses them on Citeseer, CS, and Physics by 0.3% to 2.6%. However, on four other datasets, our trainless method trails slightly. Among these, Computers and Photo exhibit a weak quasi-orthogonal property, while the OGB datasets have more training labels relative to node attribute dimensions, affirming the importance of quasi-orthogonal property and over-parameterization in Assumption 1 for our method.

TrainlessGNN on both training and validation sets. The inclusion of validation labels is a distinct advantage of TrainlessGNN, further enhancing its performance. Specifically, on the three Planetoid and two Coauthor datasets, TrainlessGNN outperforms all baseline models significantly when fitted with both training and validation labels. Remarkably, our trainless models even exceed the performance of trained GCN/SAGE models, which possess higher expressiveness with non-linear MLPs. For the remaining datasets, including the validation set also boosts TrainlessGNN's performance, aligning it with that of trained models.

5.6.3 Trained vs Trainless weight matrix

We extend our analysis by contrasting the weight matrix obtained through our method with that learned via a standard gradient descent process under cross entropy. We illustrate the learning trajectory of **SGC** over the initial 20 epochs alongside the fitted **Trainless SGC** on the Citeseer dataset. In Figure 5.5, the training loss of SGC steadily diminishes through optimization towards a minimal point, a trend guaranteed by the convex nature of the loss function. Conversely, while Trainless SGC settles at a point with relatively higher loss, its accuracy on both training (Figure 5.6) and testing (Figure 5.7) sets achieves a level comparable to the trained model. This insight implies that attaining high accuracy does not indispensably hinge on the optimization of surrogate loss. A well-generalizable model can indeed be identified without resorting to gradient descent training.

5.6.4 Varying attribute dimensions

Recalling Assumption 1, we assume that a large attribute dimension is crucial for over-parameterization, enabling our closed-form solution to approximate the optimal point effectively. We test this by varying the attribute dimensions of node attributes in the three Planetoid datasets [28], using both BOW and TF-IDF text encodings.

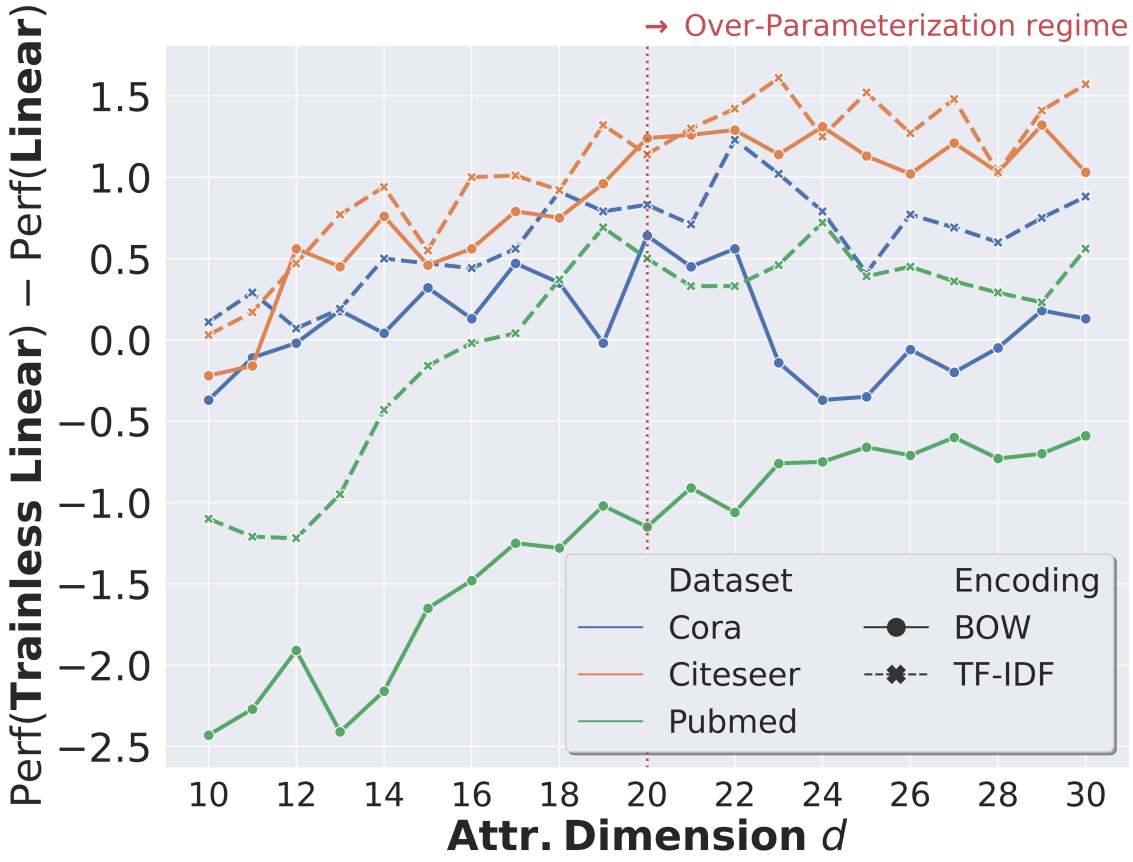

Figure 5.8: Performance comparison between **Trainless Linear** and **Linear** across varying attribute dimensions and textual encodings, with a consistent training set of 20 labeled nodes. Attribute dimensions greater than 20 (*i.e.*, $d > 20$) represent an over-parameterization regime.

We compare the performance of a **Trainless Linear** and a trained logistic regression (**Linear**) across these encodings. The results, presented in Figure 5.8, indicate that increasing attribute dimensions enhances the performance of the **Trainless Linear** model over the trained one. This supports the effectiveness of our trainless approach in semi-supervised node classification with sparse labels and lengthy text-encoded node attributes.

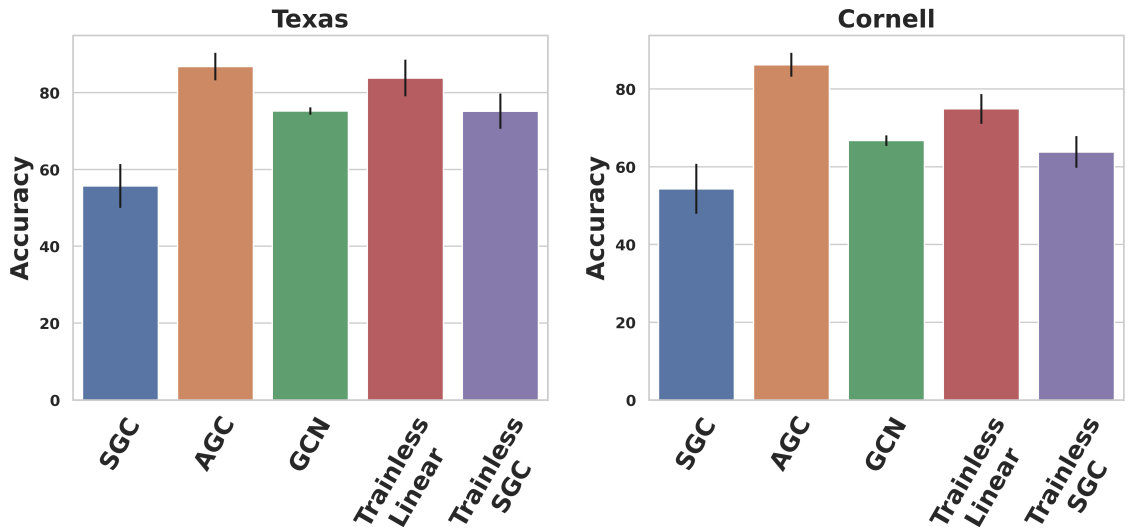

Figure 5.9: Performance of our methods on heterophilous graphs.

5.6.5 Beyond homophilous graphs

Typical GNNs like **GCN** and **SGC** generally assume graph homophily, where nodes predominantly link to similar nodes [113]. However, this isn't always the case. To assess our trainless models' effectiveness on heterophilous graphs, we experiment with two such graphs: Texas and Cornell [29]. We include **AGC** [23], a baseline model designed for heterophilous graphs, for comparison. As Figure 5.9 illustrates, our model, TrainlessGNN, not only surpasses **GCN** and **SGC** but also delivers performance on par with **AGC**. This demonstrates TrainlessGNN's adaptability to both homo/heterophilous graph structures.

5.6.6 Training efficiency

Our experiments on training efficiency show that our trainless methods are markedly faster than traditional gradient descent optimization. Owing to its one-step computation, TrainlessGNN are up to two orders of magnitude quicker than conventionally trained GNNs, including GCN and SGC. Detailed comparisons are provided in Figure

D.1 in the Appendix.

5.7 Conclusion

In this study, we ventured into an alternative approach to fitting the GNN model for addressing the semi-supervised node classification problem on TAG, bypassing the traditional gradient descent training process. We analyze the distinctive challenges inherent to semi-supervised node classification and investigate the training dynamics of GNNs on text-attributed graphs. Subsequently, we introduce TrainlessGNN, a novel method capable of fitting a linear GNN without resorting to the gradient descent training procedure. Our comprehensive experimental evaluations show that our trainless models can either align with or even outperform their traditionally trained counterparts.

PART III

ADAPTABLE GNNS

CHAPTER 6

ADAPTING PRETRAINED GNNS TO NEW GRAPH

6.1 Overview

This chapter addresses the adaptability of GNNs by proposing a universal link prediction framework (UniLP), designed to transfer knowledge learned from previously encountered graphs and adapt dynamically to new graphs during inference. Traditional GNNs are usually trained individually for each graph, which significantly limits their ability to generalize across different graph datasets—particularly when the target graph has limited training data. To overcome this limitation, we introduce UniLP, which directly encodes link representations at inference time using the unique connectivity patterns present in each target graph. This approach leverages an attention mechanism to dynamically adjust representations, effectively capturing graph-specific structural contexts without requiring additional training. Extensive experiments confirm that UniLP achieves competitive or superior performance across diverse graph datasets compared to traditional methods that require dedicated training. It demonstrates that UniLP significantly enhances GNNs adaptability and broadens their practical application scope.

This chapter primarily builds upon a pre-print manuscript [41] in collaboration with Haitao Mao, Zhichun Guo, and Nitesh V. Chawla.

6.2 Introduction

Graph-structured data is ubiquitous across diverse domains, including social networks [91], protein-protein interactions [141], movie recommendations [85], and citation networks [169]. It encapsulates the complex relationships among entities, serving as a powerful data structure for analytical exploration. At the heart of graph analysis lies the task of link prediction (LP) [37, 58, 168], a crucial problem aimed at forecasting missing or future connections within these networks. Over the years, the quest to enhance LP accuracy has advanced the development of numerous methodologies [87], broadly categorized into two main classes of approaches.

The first line of works is non-parametric heuristics link predictors, including Common Neighbor (CN) [91], Preferential Attachment (PA) [9], Resource Allocation (RA) [191] and Katz index [78]. By discovering and abstracting the universal structural properties underlying different graphs [9, 60, 158], heuristics methods are developed based on observing the connectivity patterns existing in real-world graph datasets. For example, CN assumes the tendency of triadic closure [45], such that a friend's friend is likely to be friends in a social network. These heuristics link predictors can be readily applied to any graph dataset with great generalizability. However, this approach relies on predefined heuristics, crafted from human expertise into the graph connectivity. Despite the initial success via capturing one specific connectivity pattern, they fail to capture all the effective structural features in the link prediction, leading to suboptimal performance when applied indiscriminately.

The other line of works is parametric link predictors, which automatically learn the connectivity patterns by fitting the LP models to the target graphs. These parametric methods, especially those Graph Neural Networks [59, 82] for Link Prediction (GNN4LP), have dominated the leaderboard of the link prediction tasks [62]. Typically, these GNN4LP are provably the most expressive models such that the link representation is permutation-invariant [182]. They can capture more effective

structural features compared to the simpler heuristics counterparts. However, their dependency on extensive training for each new graph dataset and the necessity for hyperparameter optimization [22, 38, 155] present notable challenges for their application across diverse graph environments.

Given that (1) the heuristics methods can be readily applied to any graphs without training based on common connectivity patterns and (2) the parametric model can automatically capture the connectivity patterns by fitting on the graph, a natural question arises:

Can a singular LP model automatically learn and apply the connectivity pattern across new, unseen graphs without the need for direct training?

An affirmative response would not only pioneer a new frontier in graph machine learning but also align with the transformative potential observed in foundation models across text and image processing fields [18, 83]. These models' exceptional generalizability, driven by their capability of transfer learning [173], offers a blueprint for the development of a universal LP model capable of broad applicability without explicit fitting.

Present work. In this study, we introduce the Universal Link Predictor (UniLP), a novel model designed for immediate application across diverse graph environments[1] without the prerequisite of model fitting. Our investigation starts by assessing whether existing LP models possess the capability to transfer connectivity pattern knowledge from one graph to another. Through empirical and theoretical analyses spanning both heuristic and parametric link predictors, we uncover a significant challenge: negative transfer [152] can happen when directly transferring the connectivity

[1]In this study, we focus on non-attributed graphs. This choice is informed by previous findings indicating that node attributes have minimal impact on the effectiveness of LP tasks [38].

patterns across distinct graph datasets, including both real-world and synthetic examples. This complexity arises from the inherent diversity and flexibility of graph data, leading to unique connectivity patterns for each graph.

To equip UniLP with the capability to adapt to diverse graphs without the need for training, we are inspired by the concept of In-context Learning (ICL) as utilized by large language models (LLMs) [18]. ICL enables models to adapt to new datasets or tasks through the relevant demonstration examples [153]. Analogously, for adapting our LP model to a particular graph, we select a collection of in-context links to act as such demonstration examples. These in-context links not only provide a context for link prediction but also aid in capturing the unique connectivity pattern inherent to the graph in question. To achieve link representations that are conditioned on the graph's specific connectivity pattern, we employ an attention mechanism [16, 147]. This mechanism facilitates dynamic adjustment of link representations in response to the graph context, enabling the model to accurately reflect the unique connectivity patterns of each graph.

We have curated a diverse collection of graph datasets spanning multiple domains, providing a rich variety of connectivity patterns for benchmarking. Through extensive experiments on these datasets, we demonstrate the seamless applicability of UniLP to novel and unseen graph datasets without requiring dataset-specific fitting. Notably, UniLP, empowered with ICL, exhibits the capability to meet or even exceed the performance levels of LP models that have been pretrained and finetuned for specific target graphs. This achievement underscores UniLP's broad applicability and robust adaptability, establishing a groundbreaking approach to link prediction tasks.

In summary, our contributions to the field of link prediction are:

- We pioneer in highlighting the challenges of applying a singular LP model across various graph datasets due to conflicting connectivity patterns, a finding supported by both empirical evidence and theoretical analysis.
- Addressing these challenges, we introduce UniLP, a novel LP approach leverag-

ing ICL for dynamic adaptation to new graphs in real-time, thereby eliminating the need for traditional training processes.

- The diverse collection of graph datasets we've collected facilitates extensive validation of UniLP's adaptability. Our experiments confirm that UniLP is not only capable of adjusting to any new graph dataset during inference but also achieves competitive performance, marking a significant advancement in link prediction methodologies.

6.3 Can one model fit all?

Machine learning models perform a task by learning from data. The quest for generalizability in machine learning models has led to significant advancement in domains such as natural language processing (NLP) [18, 147] and computer vision (CV) [12, 83]. Foundation models in these fields have demonstrated remarkable generalizability across unseen datasets [36], primarily due to their training on extensive data, which enables them to learn transferable knowledge.

In the context of LP tasks, the heuristics link predictors can be seen as a type of *transferable knowledge*. These predictors, crafted by manually analyzing common connectivity patterns in real-world graphs, offer insights into the underlying structure of networks. However, the validity of applying these heuristics universally is questioned, especially considering the wide spectrum of graph data. For instance, social networks like Facebook often exhibit a community-oriented structure [111]. Conversely, networks adhering to a scale-free power-law distribution [9], such as the World Wide Web, tend to favor a Preferential Attachment connectivity pattern. Through both empirical and theoretical examination, we aim to explore the challenges posed by the direct application of connectivity patterns from one graph to another. Our findings will reveal that such an approach may lead to negative transfer [152], emphasizing the critical need for adaptable strategies in the face of graph diversity.

6.3.1 Empirical evaluation on transferability

Our exploration begins with an empirical investigation aimed at understanding the transferability of learned connectivity patterns across diverse graph domains. We curate a collection of real-world graphs from varied fields in Table E.1, ensuring comprehensive illustrations of different graph types.

To assess the potential of the important connectivity pattern, learned from one graph, to influence the LP performance on another, we incorporate extra graphs into the training phase of the target graph. In other words, this experiment deviates from the standard supervised learning approach by introducing additional training signals from other graphs. If the connectivity patterns from these extra graphs align with or augment the structure of the target graph, the LP model's performance should either remain stable or improve. To make the experiment tractable, we only introduce **one additional** graph into the training graph and then make a link prediction on the target graph. This additional graph is kept disconnected from the target graph to ensure that the test set remains the same as standard LP tasks. . In these experiments, we employ SEAL [178] as the backbone model for the experiment and adopt Hits@50 as the performance metrics [62].

Results are presented in Figure 6.1. It shows how the LP model's performance is affected by the introduction of additional graph data during training. In the heatmap, warm colors represent an improvement in LP performance, while cooler colors denote a performance decrease. The predominance of cooler colors in the heatmap reveals that integrating an extra graph into training generally results in performance degradation. This observation underscores the potential discordance in the underlying characteristics of different graphs, leading to conflicts between the learned connectivity patterns . This phenomenon highlights the inherent challenge in deploying a singular LP model across various graphs, thus questioning the feasibility of a "one model fits all" approach in the context of LP tasks.

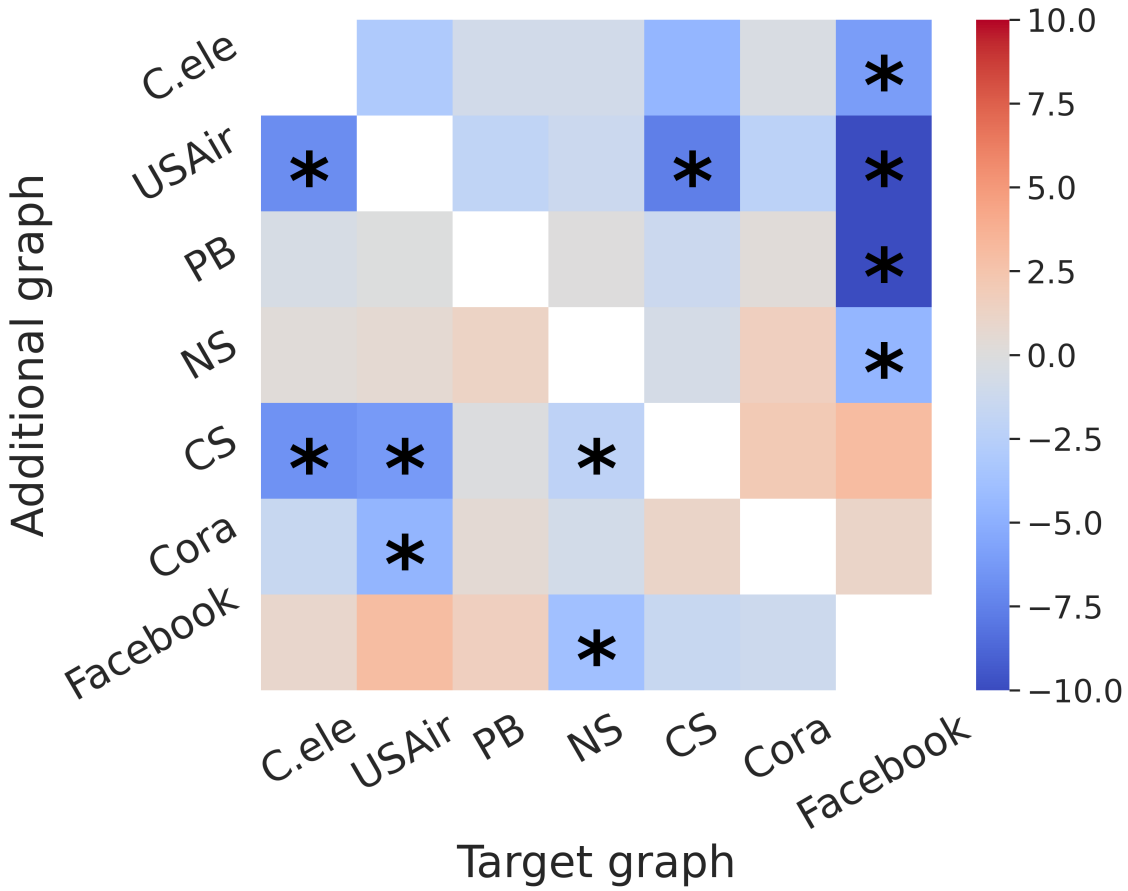

Figure 6.1: Performance change of SEAL [178] after training with one additional graph. ✱denotes statistically significant change.

6.3.2 Conflicting patterns across graphs

In this section, we delve into the theoretical aspects of how the unique characteristics of different graphs can hinder the transferability of connectivity patterns. We begin with a formal definition and preliminary discussion of LP.

Preliminary. Consider an undirected graph $G = (V, E^o)$. V is the set of nodes with size n, which can be indexed as $\{i\}_{i=1}^{n}$. E^o denotes the *observed* set of links, which is a subset $E^o \subseteq E^*$ of the complete set of true links $E^* \subseteq V \times V$. Here, E^* encompasses not only the observed links but also potential links that are currently absent or may

form in the future within the graph G. For any node $v \in V$, $\mathcal{N}(v) = \{u|(u,v) \in E^o\}$ denotes the neighbors of node v. The set of k-hop simple paths from node u to v is denoted as $\pi_k(u,v) = \{(v_1, v_2, \ldots, v_k)|v_1 = u, v_k = v \text{ and } (v_i, v_{i+1}) \in E^o \text{ for } i \in \{1, \ldots, k-1\}\}$. Note that paths only contain distinct nodes. We denote the shortest-path between a node pair (u,v) as $\text{SP}(u,v)$.

The objective of LP tasks is to identify the set of unobserved true links $E^u \subseteq E^* \backslash E^o$ within a given graph G. This task diverges from typical binary classification problems, as the potential candidates for E^u are predetermined: they consist of all node pairs not already included in the observed links $V \times V \backslash E^o$. In practical terms, "identifying" E^u equates to ranking these unobserved true links higher than false links based on their link features [62, 168]. This ranking process is defined by what we term *connectivity patterns*[2]:

Definition 5. ***Connectivity pattern*** *is an ordered sequence of events* $\omega = [A_1, A_2, \ldots]$ *such that* $p(y = 1|A_i) \geq p(y = 1|A_j)$ *for any* $i < j$.

Here, an event A refers to a specific set of conditions met by the link features of a node pair. In LP tasks, connectivity patterns may be determined by human experts using heuristic methods or by training parametric link predictors. For example, in social networks, a simple connectivity pattern might be $\omega = [CN(u,v) \geq 1, CN(u,v) = 0]$, suggesting that pairs of users with common friends are more likely to connect than those without any.

The ability to transfer a connectivity pattern from one graph to another suggests the potential for LP models to be applicable to new, previously unseen graphs. However, a mismatch in the ranking of connectivity patterns between the training and target graphs could lead to inaccuracies, since the model can assign higher scores to

[2]We have an in-depth analysis on how connectivity patterns differ from and relate to graph distributions in Appendix E.2.1, where we illustrate that graphs with different underlying distribution could have the shared connectivity pattern.

unlikely links and lower scores to likely ones.

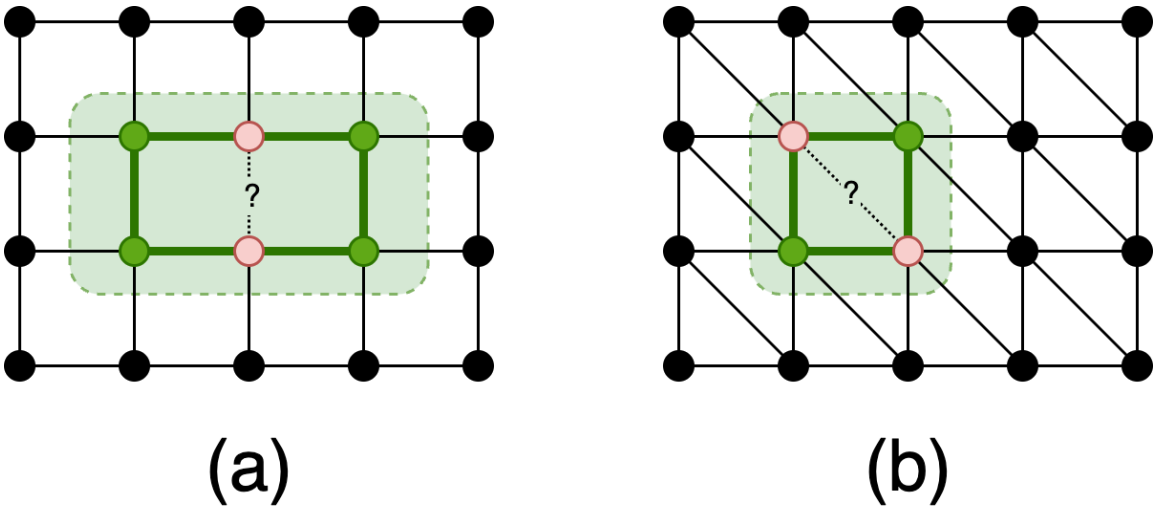

Figure 6.2: Two synthetic graphs with different connectivity patterns: (a) **Grid** lattice graph; (b) **Triangular** lattice graph.

Next, we demonstrate that even structurally similar synthetic graphs can exhibit different connectivity patterns. We begin by considering two types of lattice graphs: a **Grid** graph, similar to a chessboard, where nodes are evenly spaced on a 2D grid, each connected to its four nearest neighbors; and a **Triangular** graph, derived from the Grid by adding one diagonal edge within each square unit. Despite their structural similarities, these graphs, Grid and Triangular, display divergent connectivity patterns:

Theorem 7. *Define* $A_2 = |\pi_2(u, v)| \geq 1$ *and* $A_3 = |\pi_3(u, v)| \geq 1$ *as elements of* ω. *The connectivity patterns on Grid and Triangular graphs are distinct. Specifically: (i) On Grid:* $\omega = [A_3, A_2]$*; (ii) On Triangular:* $\omega = [A_2, A_3]$.

The proof is in Appendix 7. In essence, in Triangular graphs, node pairs two hops

away are more likely to form a link compared to those three hops away. Conversely, in Grid graphs, despite their structural similarity to Triangular graphs, node pairs two hops away have no likelihood of linking.

This observation of conflicting connectivity patterns across similar graphs underlines the challenges in knowledge transfer for LP tasks. Even slight structural variations in graphs can significantly alter the likelihood of link formation between nodes. Consequently, the task of developing a universal link predictor, capable of adapting to any graph without specific tuning for its connectivity pattern, is a non-trivial endeavor.

6.3.3 Contextualizing Link Prediction

The challenge of conflicting connectivity patterns across different graphs highlights a critical issue: a model trained on one graph may break down when applied to another without accommodating the unique characteristics of the target graph. To mitigate this, we suggest a paradigm where the model dynamically adapts to the target graph by taking into account its specific characteristics.

This adjustment process involves conditioning the model on the target graph's properties, thereby ensuring that the prediction of link formation, $p(1|A)$, is influenced not just by the inherent link features but also by the properties of the target graph. We draw inspiration from the concept of In-context Learning (ICL) in LLMs [31], which enables LLMs to solve tasks with a few demonstration examples. We propose the incorporation of the target graph as a contextual element c in the link prediction $p(1|A, c)$. By doing so, the model learns to understand the joint distribution of link features and the graph context, allowing it to adapt to different graphs. In the subsequent section, we will delve into the practical implementation of an LP model equipped with ICL capabilities.

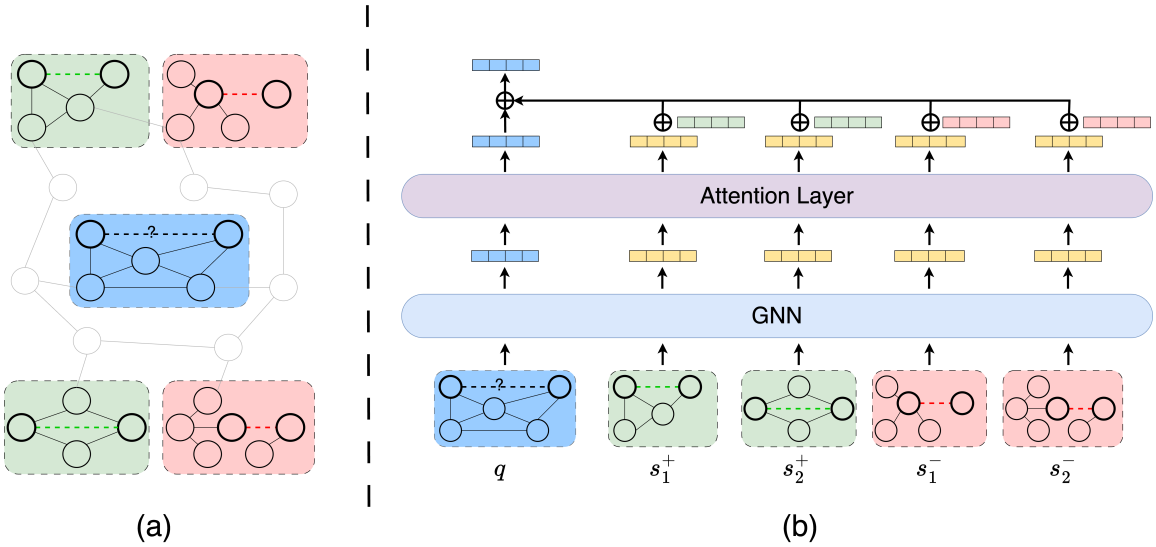

Figure 6.3: Overview of the Universal Link Predictor framework. (a) For predicting a query link q, we initially sample positive (s^+) and negative (s^-) in-context links from the target graph. Both the query link and these in-context links are independently processed through a shared subgraph GNN encoder. An attention mechanism then calculates scores based on the similarity between the query link and the in-context links. (b) The final representation of the query link, contextualized by the target graph, is obtained through a weighted summation, which combines the representations of the in-context links with their respective labels.

6.4 Universal Link Predictor

This section outlines our proposed UniLP, designed for effective application to unseen datasets. UniLP operates by first sampling a set of in-context links from the target graph, which are then independently encoded alongside the target link using a shared GNN encoder. An attention mechanism is employed to merge the representations of these in-context links in relation to their interaction with the target link, forming a composite representation for the final prediction. The overall framework is in Figure 6.3.

6.4.1 Query and in-context links

For a given target link $q \in V \times V$ in graph G, we define it as the *query* link. To predict this link based on the contextual information of G, we start by sampling a set of *in-context* links from G. Specifically, we select k node pairs $S^+ \subseteq E^o$ as positive examples, where $S^+ = \{s_1^+, s_2^+, \ldots, s_k^+\}$. These pairs have existing links between them in G. Similarly, we gather negative examples $S^- = \{s_1^-, s_2^-, \ldots, s_k^-\} \subseteq V \times V \backslash E^o$, comprising k node pairs without a link. The combined set $S^+ \cup S^-$ approximates the overall properties of G and provides a context c for the model to perform LP ($p(1|A, c)$) using both link features and graph context.

Once we get the query link and the in-context links, we need to obtain the structural representation for them. We start by extracting the ego-subgraph for each of them. An ego-subgraph $\mathbb{G}((u, v), r, G)$ for a node pair (u, v) is a subgraph induced by all the r-hop neighboring nodes of the nodes u and v on the graph G:

$$\mathbb{G}((u, v), r, G) = (V_s, E_s),$$

where $V_s = \{i|\mathrm{SP}(i, u) \leq r \text{ or } \mathrm{SP}(i, v) \leq r\}$ and $E_s = \{(i, j) \in E^o | i, j \in V_s\}$. For simplicity, we denote such an ego-subgraph as $\mathbb{G}(e)$ for the node pair $e = (u, v)$ when there is no ambiguity. The ego-subgraphs for the query link and the in-context links are $\{\mathbb{G}(e)|e \in \{q\} \bigcup S^+ \bigcup S^-\}$.

Utilizing ego-subgraphs to represent links offers several key advantages over using either individual node pairs or the entire graph. Firstly, an ego-subgraph provides a richer structural context than a mere node pair, encapsulating the local neighborhood structure around the link in question. This approach allows for a more detailed and informative representation of the link's local structures. Secondly, ego-subgraphs serve as an effective and computationally efficient approximation of global link features [178]. This is advantageous over the resource-intensive process of encoding the

entire graph. Lastly, representations at the subgraph level are inherently more expressive compared to node-level representations [54]. This enhanced expressiveness is crucial for capturing the structures of links and performing accurate LPs.

6.4.2 Encoding ego-subgraphs

The ego-subgraphs within the set $\{\mathbb{G}(e)|e \in \{q\} \cup S^+ \cup S^-\}$ vary in size, requiring a uniform approach to representation. We employ GNNs to encode these subgraphs into a consistent latent space.

In the absence of node features in non-attributed graphs, typical GNNs [59, 82, 167] require initial input vectors for each node. Common methods like assigning identical or random vectors meet this requirement but lack expressiveness about graph structures [90, 182]. To address this, we utilize the *labeling trick* technique, assigning each node i in $\mathbb{G}(e)$ a positional encoding based on its relative position to the target link $e = (u, v)$.

Our approach, *DRNL+*, is a variant of Double Radius Node Labeling (DRNL) [178] and Distance Encoding (DE) [90]. In DRNL, nodes i in $\mathbb{G}(e)$ are assigned integer labels as follows:

$$\text{DRNL}(i, (u, v)) = 1 + \min(d_u, d_v) + (d/2)[(d/2) + (d\%2) - 1],$$

where $d_u := \text{SP}(i, u)$, $d_v := \text{SP}(i, v)$, and $d := d_u + d_v$. However, DRNL doesn't distinguish nodes reachable to only one of the target nodes. Thus, *DRNL+* enhances this by using DE to assign a tuple of integers:

$$\text{DRNL+}(i, (u, v)) = \begin{cases} (0, d_u), & \text{if } d_v = \infty \\ (0, d_v), & \text{if } d_u = \infty \\ (\text{DRNL}(i, (u, v)), 0), & \text{otherwise} \end{cases} \tag{6.1}$$

After the labeling trick indicates relative positions, we apply the SAGE [59] with mean aggregation to update node representations. The final subgraph representation, $h_e \in \mathbb{R}^F$ for each link $e \in \{q\} \bigcup S^+ \bigcup S^-$, is derived by average pooling the representations of all nodes in $\mathbb{G}(e)$. We find that mean aggregation and pooling work best for such a universal link predictor, hypothesizing that this approach better accommodates varying graph sizes and node degrees, thereby enhancing model generalizability.

6.4.3 Link prediction with context

Once the ego-subgraphs are encoded into latent space, we utilize these representations $\{h_e | e \in \{q\} \cup S^+ \cup S^-\}$ to parameterize our link predictor $p(1|A, c)$ via an attention mechanism [147].

The attention scores a between the query link representation h_q and each in-context link representation h_s for $s \in S^+ \cup S^-$ are calculated using additive attention [16, 112]:

$$a_s = p^\top \text{LeakyReLU}\left(\boldsymbol{W}_k \cdot [h_q \| h_s]\right), \tag{6.2}$$

where $p^\top \in \mathbb{R}^{F'}$ is a learnable vector and $\boldsymbol{W}_k \in \mathbb{R}^{F' \times 2F}$ is a projection matrix. The concatenation operation is denoted by $\|$. The normalized attention scores α are obtained as follows:

$$\alpha_s = \text{softmax}\left(a_s\right) = \frac{\exp\left(a_s\right)}{\sum_{e \in S^+ \bigcup S^-} \exp\left(a_e\right)}. \tag{6.3}$$

We denote the attention score between the query link and a positive in-context link $s^+ \in S^+$ as α^+, and with a negative in-context link as α^-. Like in Transformer and GAT models [16, 149], multi-head attention can also be employed to capture diverse interactions between graph structures.

Remark. The attention scores are pivotal for shaping the query link's representation in the context of the target graph. We intentionally exclude label information from the attention score computation to avoid biasing the model towards easy predictions during training. This approach aligns with an "unsupervised" learning strategy, as opposed to a "supervised" one, where label information might lead the model to rely excessively on seen patterns, thus turning the attention mechanism into a *de facto* classifier. This could hinder the model's ability to generalize and adapt across varying graph structures, increasing the risk of overfitting to specific connectivity patterns not applicable to new, unseen graphs. Our empirical findings support this methodology, demonstrating that keeping the attention computation label-free significantly boosts the model's generalizability.

After the normalized attention scores α are determined, we compute the final representation for the query link q. This is achieved by applying a weighted sum to the representations of the in-context links, using the attention scores as weights. Additionally, we integrate label information into the in-context links' representations by adding corresponding learnable vectors. Formally, the final representation is calculated as follows:

$$\tilde{h}_q = \sum\nolimits_{s \in S^+} \alpha_s^+ \boldsymbol{W}_v \left(h_s + l^+\right) + \sum\nolimits_{s \in S^-} \alpha_s^- \boldsymbol{W}_v \left(h_s + l^-\right), \tag{6.4}$$

where $l^+, l^- \in \mathbb{R}^{F'}$ are learnable vectors for labels, and $\boldsymbol{W}_v \in \mathbb{R}^{F' \times F}$ is a value projection matrix. The representation $\tilde{h}_q$ encapsulates both the link features of the query link q and an estimation of the target graph G, and is then input into an MLP classifier to produce the link prediction result:

$$p(1|A, c) = \sigma\left(\mathrm{MLP}\left(\tilde{h}_q\right)\right), \tag{6.5}$$

where $\sigma(\cdot)$ denotes a sigmoid function.

6.4.4 Pretraining objective

The pretraining objective for UniLP focuses on predicting the query link q based on its own features and the context of the graph it is part of. We align this objective with standard binary classification as seen in typical parametric link prediction algorithms [22, 38, 178]. In this setting, the classification label y_e for an edge e is set to 1 if e is among the observed links E^o; otherwise, y_e is 0. Additionally, we consider a set of pretrain graphs $\mathcal{G}$, with each graph G being a member of this set. The overall pretraining loss is then defined as:

$$\mathcal{L} = \mathbb{E}_{G\in\mathcal{G}, e\in V\times V}\text{BCE}\left(\text{MLP}\left(\tilde{h}_e\right), y_e\right). \tag{6.6}$$

This loss function is employed across multiple graphs, allowing UniLP to learn a generalizable pattern for link prediction across various graph structures.

6.5 Related work

Link prediction. Traditional LP methods are handcrafted heuristics designed by observing the connectivity pattern in real-world data. They leverage either the link's local [4, 9, 91, 191] or global information [78, 116] to infer the missing links in the graph. OLP [55] stacks the heuristics link predictors as a feature vector and fits a random forest as the classifier. WLNM [177] is one of the pioneers in training a neural network as a link predictor. GAE [81], as the first GNN4LP, utilizes GNNs to encode the graph structure into node representation and perform the link prediction task. SEAL [178, 182] points out that a link-level representation is necessary for a successful LP method and proposes the labeling trick to enable GNNs to learn the joint structural representation. ELPH [22], NCNC [155], and MPLP [38] further improve the scalability of GNN4LP and achieve the state-of-the-arts on various graph benchmarks.

In-context Learning. The remarkable efficacy of LLMs across a broad spectrum of language tasks is significantly attributed to their adeptness in ICL [18]. This capability allows LLMs to generalize to new tasks by leveraging demonstration examples, effectively learning the required skills on the fly. Irie et al. [68] delves into the equivalence between conventional model training and the application of attention mechanisms to training samples during inference, suggesting an underlying mechanism of ICL. Further exploration by Dai et al. [31] posits that ICL facilitates an implicit optimization process guided by in-context examples. While the concept of ICL has been primarily associated with LLMs, Prodigy [65] represents an initial attempt to adapt ICL for GNN-based models. Their approach, however, is somewhat constrained by the overlap in pretrain and test datasets, which raises questions about the method's transferability across distinct graph domains.

6.6 Experiments

In this section, we conduct extensive experiments to assess the performance of UniLP on new unseen datasets.

6.6.1 Experimental setup

Benchmark datasets. The foundation for our model's training is a collection of graph datasets spanning a variety of domains. Following [97], we have carefully selected graph data from fields such as biology [150, 158, 180], transport [10, 158], web [3, 5, 136], academia collaboration [110, 131], citation [169], and social networks [128]. This diverse selection ensures that we can pretrain and evaluate the LP model based on a wide array of connectivity patterns. The details of the curated graph datasets can be found in Table E.1 in Appendix.

Baseline Methods. We compare UniLP with both heuristic and GNN-based parametric link predictors. Heuristic methods include Common Neighbor (CN) [91], Adamic-Adar index (AA) [4], Resource Allocation (RA) [191], Preferential Attachment (PA) [9], Shortest-Path (SP), and Katz index (Katz) [78]. GNN-based methods include GAE [81], SEAL [178], ELPH [22], NCNC [155], and MPLP [38]. For GAE and NCNC, which require initial node features, we use a 32-dimensional all-one vector. All other methods can handle non-attributed graphs directly.

Evaluation of UniLP. To evaluate UniLP's effectiveness on unseen datasets, we divide our graph data into non-overlapping pretrain and testing sets (see Table E.1) and pretrain one single model on the combined pretrain datasets. During pretraining, we dynamically sample 40 positive and negative links as in-context links $S^+ \cup S^-$ for each query link from the corresponding pretrain dataset. For evaluation, each test dataset is split into 70%/10%/20% for training/validation/testing. The training set here forms the observed links E^o, while validation and test sets represent unobserved links E^u. During the inference, we sample 200 positive and negative links as in-context links per test dataset. We report Hits@50 [62] as the evaluation metric for LP. More details about the pretraining of UniLP can be found in Appendix E.1.2.

Evaluation of Baselines. Baseline models follow similar evaluation procedures, with adaptations for transfer learning capabilities. We employ two settings: (1) **Pretrain Only**, where models are trained on combined pretrain datasets and then tested on each test dataset, and (2) **Pretrain & Finetune**, where after pretraining, models are additionally finetuned on each test dataset with 200 sampled positive and negative links for training.

6.6.2 Primary results

TABLE 6.1

LINK PREDICTION RESULTS.

	Biology	Transport	Web	Collaboration		Citation	Social	
	C.ele	**USAir**	**PB**	**NS**	**CS**	**Cora**	**Facebook**	**Ave. Rank**
Heuristics								
CN	46.88±12.28	82.75±1.54	41.15±3.77	74.03±1.59	56.84±15.56	33.85±0.93	58.70±0.35	11.00
AA	61.07±5.16	**86.96±2.24**	44.12±3.36	74.03±1.59	**68.22±1.08**	33.85±0.93	**67.80±2.12**	5.71
RA	62.80±4.84	**87.27±1.89**	43.72±2.86	74.03±1.59	68.21±1.08	33.85±0.93	**68.84±2.03**	5.57
PA	43.85±4.12	77.69±2.29	28.93±1.91	35.35±3.01	6.49±0.61	22.09±1.52	12.95±0.63	13.71
SP	0.00±0.00	0.00±0.00	0.00±0.00	80.00±1.11	41.34±35.58	52.97±1.53	0.00±0.00	13.14
Katz	58.86±6.48	84.64±1.86	44.36±3.65	78.96±1.35	66.32±5.59	52.97±1.53	60.79±0.60	6.86
Pretrain Only								
SEAL	61.28±3.76	86.00±1.56	45.44±2.68	84.07±1.96	62.82±1.62	56.21±2.24	54.57±1.48	5.57
GAE	44.71±3.39	76.12±2.27	27.56±2.34	15.20±2.14	5.08±0.48	24.22±1.53	6.65±0.57	14.14
ELPH	59.23±4.50	84.42±2.22	43.69±2.90	84.27±1.43	**70.69±3.63**	56.91±1.43	61.80±2.46	5.57
NCNC	48.07±4.79	75.44±4.22	25.66±1.42	80.07±1.43	34.27±2.28	52.51±2.54	19.28±1.58	11.57
MPLP	56.74±5.31	82.94±2.30	**47.78±2.55**	80.33±1.54	24.26±1.28	46.71±2.25	48.06±2.09	8.86
Pretrain & Finetune								
SEAL	**64.45±4.14**	**88.49±2.16**	47.78±3.32	84.84±2.32	61.54±3.09	**62.19±3.27**	58.70±2.78	**3.14**
GAE	44.71±4.07	74.47±2.96	25.92±2.64	18.34±2.27	4.95±0.44	25.31±1.48	6.11±0.39	14.71
ELPH	60.51±5.72	84.52±2.08	43.58±3.48	**86.08±0.69**	**71.10±3.48**	57.18±1.89	63.31±3.63	4.71
NCNC	**64.45±5.10**	85.85±2.46	47.75±6.90	**88.25±2.25**	58.75±5.74	**60.00±2.50**	59.32±6.93	**3.71**
MPLP	62.56±4.79	85.08±1.54	**48.01±2.94**	80.16±1.18	50.35±1.01	56.02±2.09	56.72±1.20	6.14
Ours								
UniLP	**65.20±4.40**	85.98±2.00	**48.14±2.99**	**89.09±2.05**	64.59±2.65	**57.50±2.40**	**65.49±2.05**	**1.86**

Link prediction results on test datasets evaluated by Hits@50. The format is average score ± standard deviation. The top three models are colored by **First**, **Second**, **Third**.

Table 6.1 presents the performance of UniLP on various unseen graph datasets. The results demonstrate that UniLP outperforms both traditional heuristic methods and standard GNN-based LP models that are pretrained without specific adaptation, showing significant improvements in 4 out of 7 the benchmark datasets. This performance enhancement suggests that tailoring the LP model to individual graphs can markedly increase its transfer learning capabilities.

Moreover, UniLP achieves comparable or even superior results to GNN-based LP models that undergo finetuning, despite not being explicitly trained on the test data. This highlights the effectiveness of the ICL capability in UniLP, which allows the model to adapt seamlessly to specific graph datasets without the need for additional training. By leveraging in-context links provided during the inference phase, UniLP can dynamically adjust its knowledge of connectivity patterns, demonstrating its potential to deliver robust performance across a wide range of unseen graph datasets. In addition, the experimental results on the synthetic Triangular/Grid lattice graphs can be found in Table 6.3.

6.6.3 The inner mechanism of UniLP

We further explore the capability of our proposed model's ICL to facilitate skill learning [99, 117], enabling the model to acquire new skills not encountered during the pretraining phase, guided by ICL demonstrations. This investigation focuses on the model's performance sensitivity to corrupting in-context links, particularly when these links are presented with incorrect input-label associations. Given that each in-context link consists of an input and its corresponding label, we introduce two perturbation strategies to assess this sensitivity: **FlipLabel**: we invert the labels of the in-context links, labeling previously positive links as negative and vice versa. **RandomContext**: Instead of selecting in-context links from the target graph, we randomly sample them from a graph generated using the Stochastic Block Model [60].

TABLE 6.2

LINK PREDICTION RESULTS UNDER CONTEXT PERTURBATION.

	Biology	Transport	Web	Collaboration		Citation	Social
	C.ele	**USAir**	**PB**	**NS**	**CS**	**Cora**	**Facebook**
UniLP-FlipLabel	0.61±0.27	15.81±13.17	0.03±0.03	27.97±4.29	0.60±0.23	2.03±0.55	0.32±0.15
UniLP-RandomContext	52.89±5.90	81.91±2.14	47.47±3.05	85.60±1.23	47.80±6.48	37.62±5.64	22.17±6.55
UniLP	65.20±4.40	85.98±2.00	48.14±2.99	89.09±2.05	64.59±2.65	57.50±2.40	65.49±2.05

Link prediction results on test datasets evaluated by Hits@50 under context perturbation. This table presents the outcomes of link prediction when the context, i.e., in-context links, is deliberately altered. The aim is to analyze how changes in the context influence the final prediction accuracy.

The outcomes, as shown in Table 6.2, reveal that flipping the labels of in-context links significantly degrades the model's performance, rendering it almost ineffective. This finding underscores the model's utilization of ICL for skill learning, specifically in learning new feature-label mappings within a given context [106, 159]. It highlights the pivotal role of accurate label information in in-context links for the model's effective adaptation to the target graph.

Furthermore, using randomly generated graphs as a source of in-context links also detrimentally affects performance, albeit to varying extents across different datasets. This implies the importance of choosing in-context links that genuinely represent the properties of the target graph. Interestingly, the less severe performance decline in some datasets may indicate their inherent community-based graph structures.

6.6.4 Effectiveness of in-context links' size

This experiment evaluates how varying the quantity of in-context links affects UniLP's performance during inference. We experiment with different numbers of in-context links, ranging from 10 to 400, sampled from each test graph . These links are

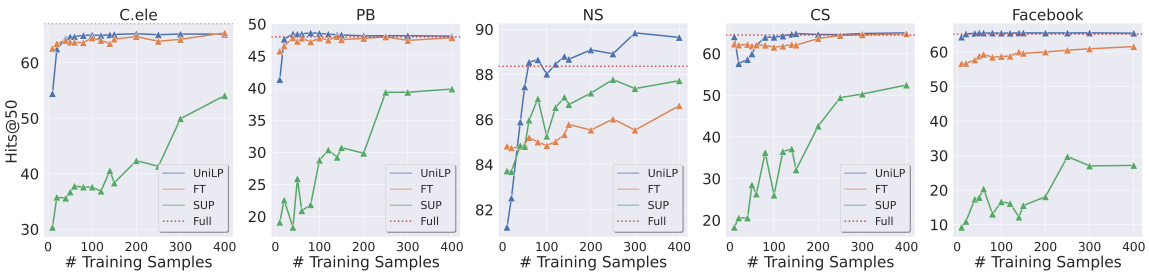

Figure 6.4: Performance of UniLP with varying quantities of in-context links.

used as context for the model. Additionally, we utilize SEAL as the base model and assess its performance under finetuning (FT) and supervised training (SUP) from scratch with varying training sample sizes. For comparison, we also include results from training a SEAL model on the full set of target graph data (Full), as detailed in Figure 6.4.

The findings reveal a consistent improvement in UniLP's performance with an increasing number of in-context links. This indicates that our method can more effectively capture the target graph's properties with additional context. Notably, on four of the test datasets, UniLP either matches or exceeds the performance of models trained end-to-end on the entire graph. This suggests that leveraging more pretrpre-train data can be advantageous for LP tasks when properly managed. Furthermore, despite both UniLP and the finetuned models being pretrained on the same datasets and using the same in-context links, UniLP occasionally outperforms its finetuned counterparts. This observation suggests that in some cases, utilizing ICL can be a more effective approach for adapting a pretrained model to a specific target dataset compared to finetuning. The trend on the rest of graphs can be found in Figure E.1.

6.6.5 Visualization of the link representation

We conduct a comparative visualization of link representations as learned by a Pretrained Only SEAL model and UniLP. This comparison is shown in Figure 6.5 and Figure 6.6. The results indicate that a naively pretrained model tends to map link representations from various graph datasets into a close subspace, potentially leading to indistinguishable link representations across different graphs, even when these graphs exhibit conflicting connectivity patterns.

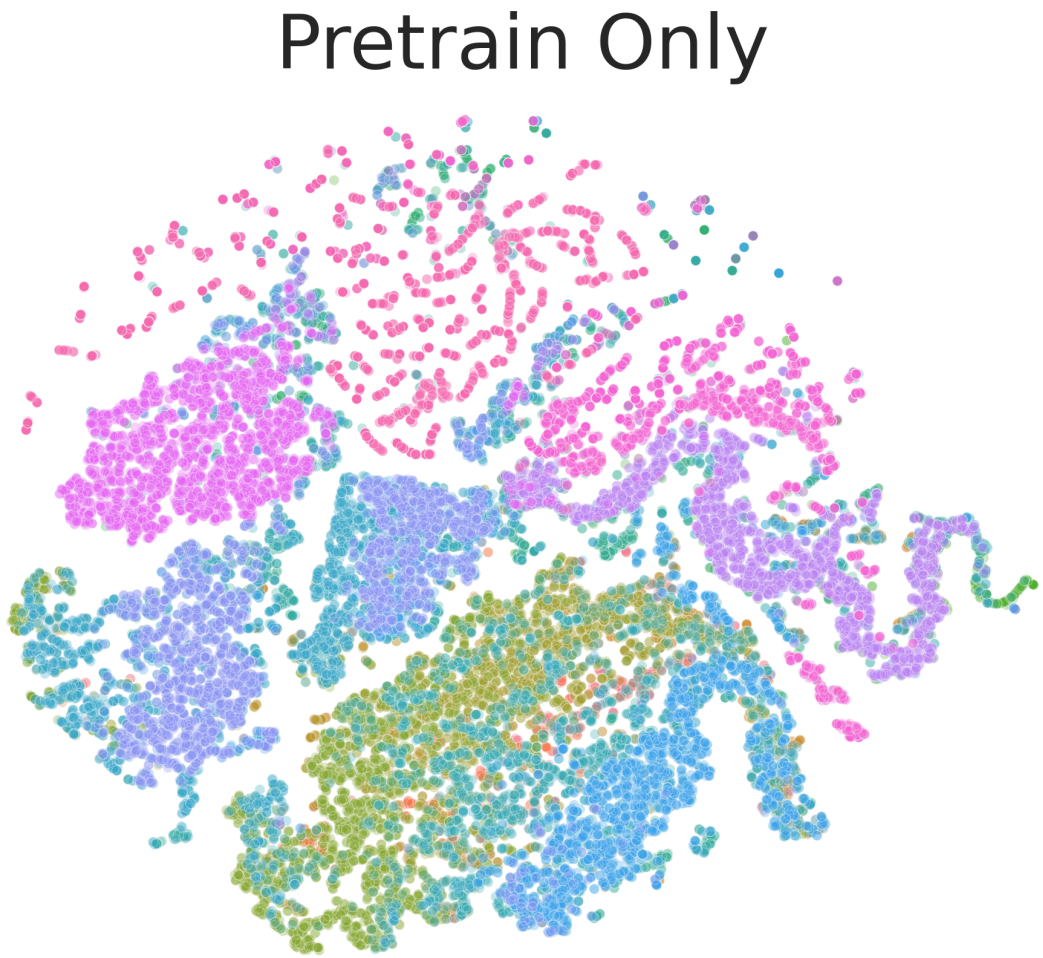

Figure 6.5: Visualization of the link representation from Pretrain Only SEAL. Different colors indicate different test datasets.

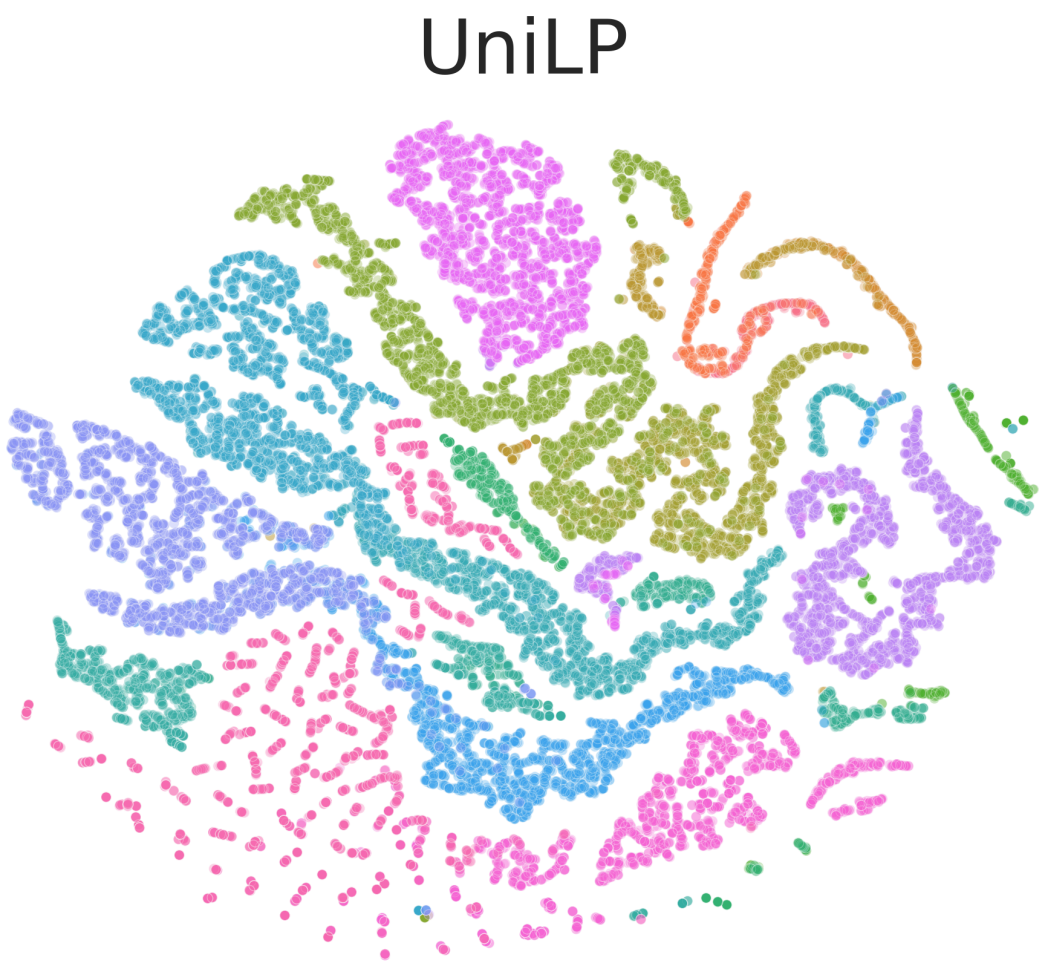

Figure 6.6: Visualization of the link representation from UniLP. Different colors indicate different test datasets.

In contrast, the link representations generated by UniLP, which are conditioned on the context of the target graph, demonstrate a distinct separation between different datasets. This separation is indicative of UniLP's effective ICL capability, which adeptly captures the subtle distributional differences across graphs. By adjusting the link representations based on the context provided by in-context links, UniLP can effectively address the challenge of conflicting connectivity patterns in diverse graph datasets.

6.6.6 Synthetic graphs

TABLE 6.3

LINK PREDICTION RESULTS ON SYNTHETIC GRAPHS.

	Triangular	Grid
Heuristics		
CN	73.58±0.81	0.00±0.00
AA	73.58±0.81	0.00±0.00
RA	73.58±0.81	0.00±0.00
PA	0.00±0.00	0.00±0.00
SP	97.91±0.63	**86.04±1.11**
Katz	90.08±0.67	56.79±0.99
SEAL		
Supervised	**99.29±0.28**	**79.45±1.09**
Pretrained Only	98.11±0.84	61.48±0.57
Pretrain & Finetune	**98.35±0.57**	**78.24±0.79**
Ours		
UniLP	**98.73±0.49**	77.39±1.38

Link prediction results on synthetic Triangular/Grid lattice graphs evaluated by Hits@50. The format is average score ± standard deviation. The top three models are colored by **First**, **Second**, **Third**.

We deployed our pretrained UniLP on the synthetic graph shown in Figure 6.2, with outcomes presented in Table 6.3. These findings demonstrate that UniLP matches the performance of both models that are fully trained on the entire graph and those that undergo explicit finetuning. This performance underscores the efficacy of UniLP's ICL capability, affirming its ability to dynamically adapt to synthetic graph

environments and learn connectivity patterns directly from in-context links without the need for additional training or finetuning.

6.6.7 Diversifying context

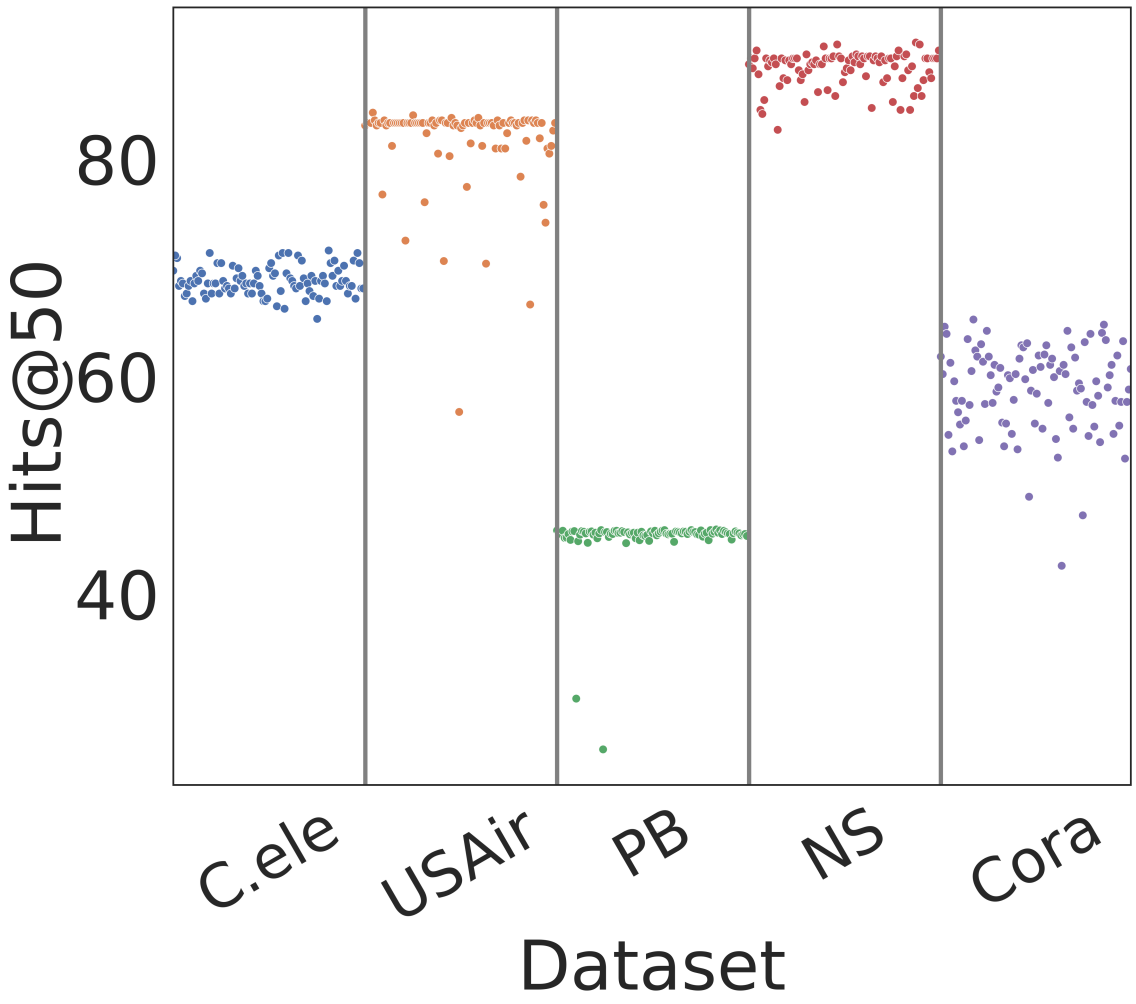

Figure 6.7: Diverse sets of in-context links on LP performance.

In our prior analysis, we utilized a set of in-context links sampled from each target graph to serve as context. This section delves into the impact of varying these in-context links by employing different random seeds, aiming to discern the sensitivity of UniLP's performance to the specific composition of in-context links for each target graph. The outcomes of this investigation are detailed in Figure 6.7. The findings reveal that the choice of in-context links indeed affects UniLP's performance across the test datasets to varying extents, highlighting the importance of the selection process for these contextual links in optimizing the model's efficacy. We leave the study on the selection of the in-context links as future work.

6.6.8 Varying positive-to-negative ratios of in-context links

In our previous exploration of LP performance, we initially maintain a balanced set of positive and negative in-context links for each query link during both pretraining and testing phases. This study delves into the effects of changeing the ratio of positive to negative in-context links, while keeping the total count constant at 200, to assess the impact on LP accuracy. The findings, shown in Figure 6.8, reveal several noteworthy observations.

Remarkably, for the majority of graph datasets examined, increasing the proportion of negative samples—contrary to intuitive expectations—does not detract from performance and, in some cases, matches the efficacy of a balanced distribution of in-context links. This phenomenon indicates that negative samples are equally informative as positive ones for leveraging the ICL capabilities of UniLP. Specifically, in the case of the synthetic Grid graph, a higher ratio of negative samples significantly enhances LP performance, given a fixed total number of in-context links. This improvement may stem from the symmetry of positive link structures within the Grid graph, which exhibit a consistent connectivity pattern. The introduction of a greater variety of negative samples seems to enrich the model's learning context, effectively

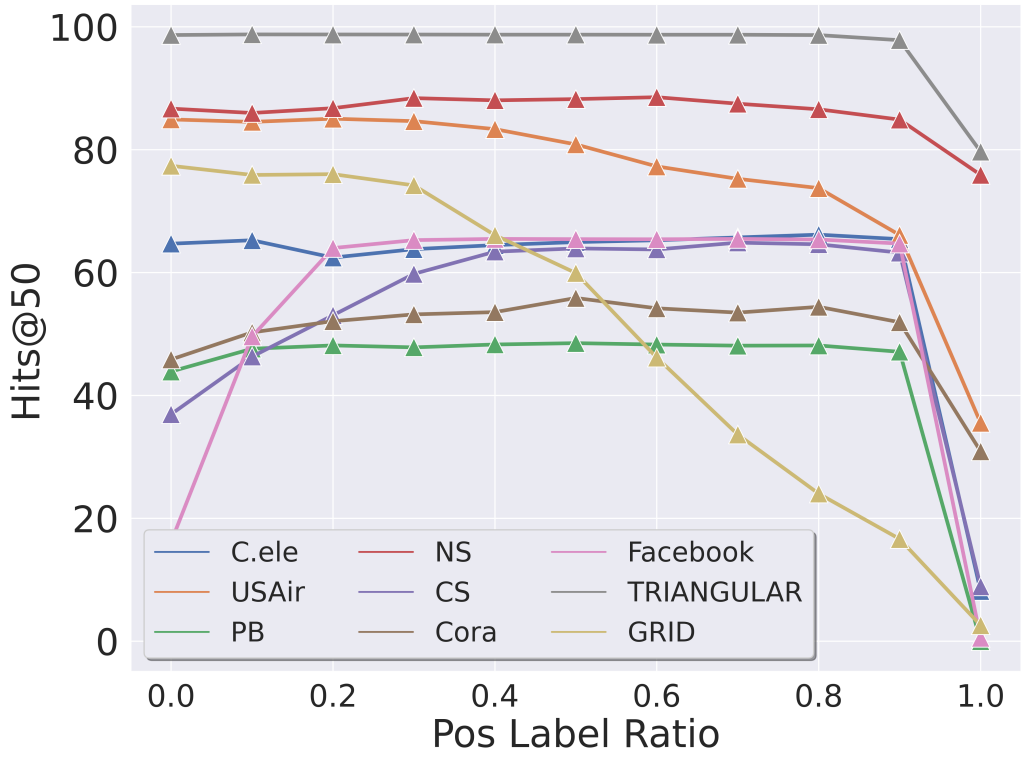

Figure 6.8: Influence of positive-to-negative in-context link ratios on LP performance.

harnessing UniLP's ICL potential to capture more diverse patterns.

An exception to this trend is observed with the Facebook graph dataset, where a balance between positive and negative in-context links yields the most favorable outcomes. This suggests that for certain graph types, a balanced approach to in-context link selection optimizes LP performance.

6.7 Conclusion

In this paper, we introduce the Universal Link Predictor, a novel approach designed to be immediately applicable to any unseen graph dataset without the ne-

cessity of training or finetuning. Recognizing the issue of conflicting connectivity patterns among diverse graph datasets, we innovatively employ ICL to dynamically adjust link representations according to the specific properties of the target graph by conditioning on support links as contextual input. Through extensive experimental evaluations, we have demonstrated the effectiveness of our method. Notably, our Universal Link Predictor excels in its ability to adapt seamlessly to new, unseen graphs, surpassing traditional models that require explicit training. This significant advancement presents a promising direction for future research and applications in the field of LP.

CHAPTER 7

ADAPTING GNNS TO RELATIONAL DATABASE

7.1 Overview

This chapter further expands the adaptability of GNNs by exploring their application in scenarios where data is not originally represented in graph form, specifically focusing on relational databases. Despite their extensive practical use in tasks such as recommender systems, fraud detection, ad click prediction, and demand forecasting, relational databases have been largely overlooked by contemporary deep learning literature. Addressing this gap, we propose transforming relational databases into unified heterogeneous graphs, enabling the direct application of end-to-end GNN models. Traditionally, tasks on relational databases require extensive manual feature engineering to consolidate useful information, increasing complexity and limiting the generalizability of model architectures. By converting relational data into graphs, we integrate feature learning and task modeling into a single step, enabling end-to-end training of GNNs. This unified approach significantly simplifies the modeling process and broadens the practical applicability of GNNs beyond conventional graph-based domains. We demonstrate the effectiveness of this strategy by applying GNNs to the task of transaction categorization in QuickBooks, achieving significant performance compared to existing baselines deployed in production.

This chapter primarily builds upon a joint work with Padmaja Jonnalagedda, Xiang Gao, Ayan Acharya, Maria Kissa, Mauricio Flores, Nitesh V. Chawla, and Kamalika Das.

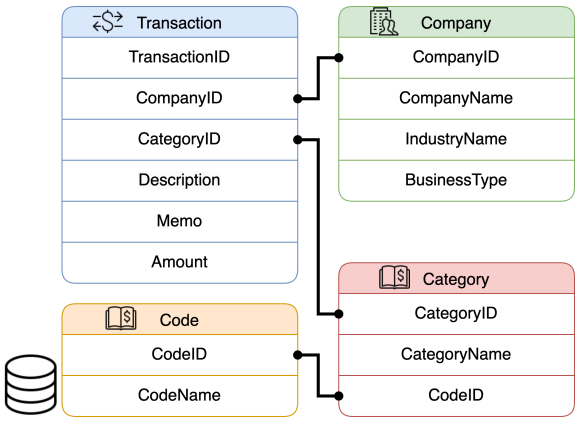

Figure 7.1: The schema of the relational database of QuickBooks transactions.

QuickBooks offers essential bookkeeping and accounting capabilities tailored to the needs of small and medium-sized businesses. It enables them to efficiently manage critical aspects of their business operations, including accounting, payroll, payments, and inventory. A key feature of QuickBooks is its ability to categorize financial activities flexibly, enhancing insights into business performance and streamlining tax compliance. By automating labor-intensive and error-prone tasks, QuickBooks allows business owners to focus on driving growth and increasing revenue.

At the core of QuickBooks's functionality is its advanced banking experience. Most business transactions today are processed through financial institutions, and QuickBooks integrates seamlessly with these institutions, enabling businesses to link their accounts and synchronize data. This connectivity triggers an influx of transactions—approximately 6.2 billion annually into QuickBooks. Having business owners or accountants manually review their transactions would be an ineffective use of their

time. Automating the processing of such a vast volume of transactions is crucial.

To facilitate its sophisticated accounting features, QuickBooks organizes transactions into specific categories or accounts. For example, a fuel purchase at an Exxon-Mobil station might be categorized under "Cars and Trucks", while an electricity bill could be classified as "Utilities". This paper addresses QuickBooks's transaction categorization challenge, employing state-of-the-art methods in natural language processing [34] and graph machine learning [17] to conceptualize this as a link prediction problem [38] within a relational database. Drawing inspiration from Relational Deep Learning [51], we propose a unified approach to effectively model the transaction database through interconnected relational tables, introducing modifications specific to the unique challenges and practical requirements of QuickBooks.

7.2.1 Problem statement

In QuickBooks, effective bookkeeping relies on the accurate categorization of transactions into specific accounts. Businesses have unique needs and preferences for how transactions are categorized. QuickBooks accommodates this by allowing customization of account names. This feature enables different companies to maintain both common and distinct account names based on their individual requirements. To further refine the organization of financial data and support compliance with tax regulations, QuickBooks encourages users to classify these account names into a structured hierarchy of more abstract account types. This system not only personalizes the accounting experience for each business but also guarantees the accurate recording of financial activities. *The more abstract account type, referred to as Code, corresponds to the IRS tax code, while the more granular account name, Category, is user-defined. For example,* Category *"Airfare" and "Internet" can both be grouped into a more abstract* Code *"Expenses".*

The primary task we address in this paper is predicting the appropriate *Cate-*

gory for any new transaction imported into QuickBooks. In addition to delivering the most likely categorization for a new transaction, we explore providing the top-5 probable Categories. This approach is predicated on the likelihood that the company's preferred *Category* is more often found within the top-5 predictions rather than solely the top prediction. Offering a selection of probable Categories can significantly enhance user trust in QuickBooks's capabilities, fostering a stronger reliance on QuickBooks for critical financial management tasks.

The data supporting this transaction categorization task is managed within a relational database, where various tables are interconnected through primary and foreign keys. The database schema, detailed in Figure 7.1, includes the following critical tables:

1. **Transaction table**: Stores records of all transactions across different companies.
2. ***Category* table**: Contains all specific account names used by QuickBooks users.
3. ***Code* table**: Includes the abstract account types aligned with overarching tax codes, that facilitate the organization of *Category*.
4. **Company table**: Contains information about companies utilizing QuickBooks.

7.2.2 Related Works

Transaction categorization is fundamental to the user experience in QuickBooks. To date, two main approaches have been developed to address this task.

The first approach, known as IRIS [88], categorizes incoming transactions based solely on a company's historical data. It begins by extracting business entity names from transaction descriptions using a rule-based normalization process that includes case folding and digit folding. IRIS then queries the company's historical transactions for similar business entities, defining similarity via Jaccard similarity—entities are

considered similar if they are frequently categorized together. A weighted voting mechanism is then employed to determine the most likely *Category* for the new transaction. While IRIS is an efficient system capable of managing large datasets and ensuring quick database queries, its reliance on a company's historical data limits its generalizability and scalability, especially for new users with limited transaction history.

The second methodology is embodied in the Shorthair and Lynx [92], currently the production model in QuickBooks. Shorthair addresses the limitations of IRIS by enhancing performance for users new to the system (cold-start users) with a populational model. It utilizes a Word2Vec-based [104] encoding for transactions and *Category*, and applies a contrastive learning framework to maximize the matching pairs of transaction-*Category* and minimize the non-matching pairs. Lynx, built on top of Shorthair, employs a logistic regression classifier to enable personalized categorization for each company. During inference, a calibrated model of Shorthair and Lynx is applied to predict the category of a new transaction. Although the Shorthair and Lynx model has demonstrated effective performance in practice within QuickBooks, maintaining an individual logistic regression classifier for each company introduces significant overhead and risks of overfitting. Additionally, calibrating two models can lead to suboptimal performance due to challenges in effectively leveraging strengths of both models.

7.2.3 Challenges and Contributions

Challenges. The transaction categorization task in QuickBooks presents several core challenges. Firstly, the transaction data is structured as a relational database, posing challenges on how to effectively model it using a unified machine learning approach. Secondly, the transaction description field, crucial for identifying the semantic meaning of a transaction, often follows the formatting standards of financial

institutions rather than natural language. This necessitates an effective method to encode transaction data, capturing nuances such as business entity names and financial acronyms. Thirdly, the vast scale and skewed distribution of *Category* labels result in a highly imbalanced learning scenario. The categorization is highly personalized so that one transaction can be put into different *Category* by different users. Traditional multi-class classifiers can struggle with effectiveness and generalizability on this task. Last but not the least, the large volume of transactions processed by QuickBooks requires a balance between model capability and computational efficiency to ensure real-time performance.

Contributions. We summarize our contributions as follows:

1. To model the relational database of the transaction data, we transform the database to a heterogeneous graph and apply a unified graph model, `Rel-Cat`, to effectively represent the relationships within the data and support both new and old users of QuickBooks. To the best of our knowledge, we are the first to apply the principles of Relational Deep Learning to a real-world problem in practice.
2. To encode the transaction data with unique linguistic characteristics, we integrate a trained-from-scratch text encoder `Txn-Bert` into `Rel-Cat`, which can effectively capture the semantics of the transaction data.
3. To handle large-scale transaction efficiently, we introduce practical techniques to improve `Rel-Cat`'s scalability, including a novel diversity filtering protocol, a similarity-based neighbor sampling, edge direction dropping, and Top-K Nearest Neighbor early exit.

7.3 `Txn-Bert`: Text Encoder trained from scratch

In this section, we introduce `Txn-Bert`, a text encoder developed specifically for encoding transaction descriptions into fixed-length embeddings. The entire encoder is trained from scratch, recognizing the unique linguistic patterns found in transaction data. We begin with the rationale behind pretraining a new language model, followed by the detailed process of the training of the transformer model.

7.3.1 Unique Linguistic Characteristics of Transaction Data

To effectively convert transaction text data into embeddings, it is essential to first examine the unique characteristics of this data.

Transaction descriptions, found in banking statements, serve as crucial identifiers for financial activities within businesses. These descriptions adhere to formatting standards set by financial institutions. For instance, Visa mandates that business entity names in transaction descriptions be no longer than 25 characters, requiring abbreviations as necessary.[1].

Such descriptions, therefore, consist predominantly of abbreviated business names, a feature markedly distinct from the more fluid and expansive natural language. This distinctiveness necessitates a specialized approach for encoding transaction. The conventional language models, typically pretrained on generic natural language datasets, do not suffice for capturing the nuances of transaction descriptions. This inadequacy forms the core motivation for developing `Txn-Bert` from the ground up, ensuring it is finely tuned to the specific lexicon and syntax of transaction language.

7.3.2 Training the tokenizer

Transaction descriptions, by their nature, resemble a constructed language distinct from natural language due to the structured format. This unique format is filled with variations of business entity names and their abbreviations, necessitating a customized vocabulary for effective text processing.

To create this specialized tokenizer, we start by gathering all available transaction descriptions to form a dedicated corpus. We employ the WordPiece tokenization method [130]—similar to the one used in the BERT model [164]—but develop a

[1]https://usa.visa.com/content/dam/VCOM/download/merchants/visa-merchant-data-standards-manual.pdf

unique vocabulary tailored to the transaction descriptions.

Transaction Description	Bert Tokenizer	Txn-Bert Tokenizer
INTUIT QUICKBOOKS	['int', '##uit', 'quick', '##books']	['intuit', 'quickbooks']
LINKEDIN JOB*XXXXXXLNKD.IN/BILL CA	['linked', '##in', 'job', '*', 'xx', '##xx', '##xx', '##ln', '##k', '##d', '.', 'in', '/', 'bill', 'ca']	['linkedin', 'job', '*', 'xxxxxx', '##lnkd', '.', 'in', '/', 'bill', 'ca']
PAYPAL *ZOOMVIDEOCO	['pay', '##pal', '*', 'zoom', '##vid', '##eo', '##co']	['paypal', '*', 'zoom', '##video', '##co']
Zelle payment	['ze', '##lle', 'payment']	['zelle', 'payment']
FACEBK MENLO PARK	['face', '##b', '##k', 'men', '##lo', 'park']	['facebk', 'menlo', 'park']
VENMO DES:CASHOUT	['ve', '##n', '##mo', 'des', ':', 'cash', '##out']	['venmo', 'des', ':', 'cashout']
AMZN Mktp US*0000Amzn.com/bill WAUS	['am', '##z', '##n', 'mk', '##tp', 'us', '*', '000', '##0', '##am', '##z', '##n', '.', 'com', '/', 'bill', 'wa', '##us']	['amzn', 'mktp', 'us', '*', '0000', '##amzn', '.', 'com', '/', 'bill', 'waus']
AMEX EPAYMENT	['am', '##ex', 'epa', '##yme', '##nt']	['amex', 'epayment']

Figure 7.2: Tokenization of transactions by BERT and `Txn-Bert`. Bert tokenizer tends to split business entity names to sub-tokens, while `Txn-Bert` tokenizer can keep them complete.

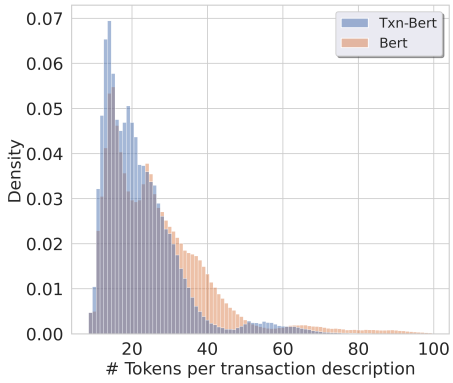

Figure 7.3: `Txn-Bert` can tokenize transactions into fewer tokens.

We compare the tokenization of transaction descriptions between the standard BERT model and our pretrained `Txn-Bert`. As illustrated in Figure 7.2, the stan-

dard BERT tokenizer often breaks down business names into smaller pieces, whether abbreviated or not. This occurs because these business names, despite being prevalent in transaction descriptions, are not included in the standard BERT vocabulary, which is designed for general natural language. In contrast, our custom-built tokenizer, which is trained specifically on transaction data, recognizes complete business names as single tokens. This capability not only simplifies the text representation but also allows for more efficient data processing, as shown in Figure 7.3. This indicates that our trained-from-scratch tokenizer can capture the entire transaction descriptions with fewer bits and represent the text distribution better.

7.3.3 Training the transformer model

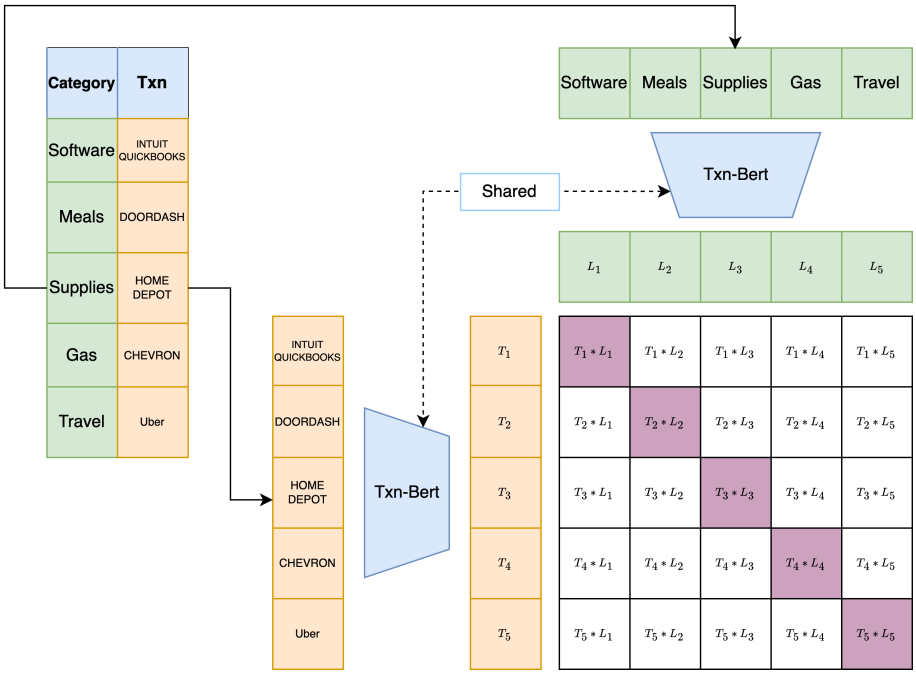

Figure 7.4: Training paradigm of `Txn-Bert`.

Once we have built a tokenizer customized for transaction descriptions, the next step is to train our text encoder from scratch. We train the model to classify new transactions into specific *Category* solely based on the information from that transaction—like its description, amount, and any additional memo—without considering the historical patterns of the company. We model our training approach after Sentence-Bert [125], treating it as a sentence pair matching problem to match transactions to *Category*. The training paradigm of `Txn-Bert` is shown in Figure 7.4. We adopt a Siamese network architecture, where a shared text encoder independently processes a pair of a transaction and its corresponding *Category*.

We format each transaction by combining its key fields into a single sentence. We include the {description}, {amount}, and {memo}. We also add a {polarity} field, which labels the transaction as "received" if the amount is positive or "paid" if it's negative. The complete transaction text looks something like this: "Transaction {polarity} ${amount} for: {description} {memo}." For the *Category* labels, we use them just as they are.

The text encoder is a transformer [147], which will refine the representation of each token iteratively through the transformer blocks. After processing through these layers, we take a mean pooling strategy to get a single representation for the whole sentence. The configuration of the transformer can be found in Appendix F.1.1.

The training objective mirrors the CLIP approach [124], optimizing for high cosine similarity between matched transaction-*Category* pairs and low similarity for unmatched pairs. We use a symmetric cross-entropy loss. More specifically, given a batch of N transaction-*Category* pairs $\{(T_i, L_i)|1 \leq i \leq N\}$, and our text encoder $f(\cdot)$, the training loss is:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} \log\left(\frac{e^{\text{Sim}(f(T_i),f(L_i))}}{\sum_{j=1}^{N} e^{\text{Sim}(f(T_i),f(L_j))}}\right) + \log\left(\frac{e^{\text{Sim}(f(T_i),f(L_i))}}{\sum_{j=1}^{N} e^{\text{Sim}(f(T_j),f(L_i))}}\right)$$

where $\text{Sim}(v, w) = \frac{v \cdot w}{|v| \cdot |w|}$ represents the cosine similarity.

This method leverages all non-matching pairs within a batch for contrastive learning, thus eliminating the need for explicit negative sampling. This effective training allows `Txn-Bert` to robustly encode transaction and *Category* data for use in downstream tasks.

7.4 `Rel-Cat`: Modeling relations within database

In this section, we introduce `Rel-Cat`. We outline our approach for preprocessing a relational database into a heterogeneous graph, thereby redefining the transaction categorization task as a link prediction problem within this graph. We then describe `Rel-Cat`, a hybrid method combining rule-based early exit (`TopK NN`) with a robust heterogeneous graph neural network (GNN). The overview pipeline can be found in Figure 7.5.

7.4.1 Build a heterogeneous graph from a relational database

The transaction data in QuickBooks is organized within a relational database comprising multiple interconnected tables, each representing different facets of the transaction data. To unify modeling across this multi-table architecture, we employ the concept of Relational Deep Learning [51], transforming the relational database into a heterogeneous graph. We provide an overview of this transformation process with a conversion diagram depicted in Figure 7.6.

7.4.1.1 Conversion overview

Referencing Figure 7.1, the relational database for transaction data consists of four key components:

1. **Multiple tables**: The database is structured into several tables, each detailing a specific aspect of the transactions.

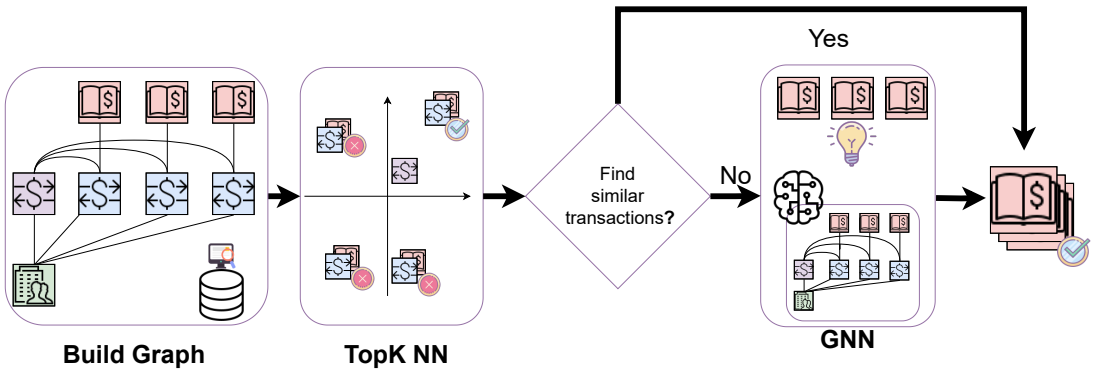

Figure 7.5: The overview pipeline of `Rel-Cat`.

2. **Rows within tables**: Each table comprises multiple rows, each row encapsulating a distinct transaction or fact.
3. **Foreign-primary key relationships**: Rows across different tables are interconnected via foreign-primary key associations, facilitating relational references among them.
4. **Row attributes**: Each row includes attributes that describe its elements, such as the transaction description in the transaction table or the company name in the company table.

These elements of the relational database are mapped to the components of a heterogeneous graph as follows:

1. **Node types**: Each table in the database is treated as a distinct node type within the graph.
2. **Nodes**: Individual rows within a table are represented as nodes with the node type corresponding to their table.
3. **Edges**: Connections between rows (nodes) across tables, facilitated by foreign-primary key pairs, are represented as edges in the graph.
4. **Node attributes**: Attributes of each row, particularly textual data, are utilized as node attributes in the graph.

7.4.2 Conversion details

Adopting the notations from [51], we define the relational database as $(\mathcal{T}, \mathcal{L})$, consisting of a collection of tables $\mathcal{T} = \{T_i\}$ where i represents different tables such as "transaction", "company", "*Code*", or "*Category*". The relationships between these tables are captured by links $\mathcal{L} \subseteq \mathcal{T} \times \mathcal{T}$. An edge $L = (T_{\text{fkey}}, T_{\text{pkey}})$ exists if a foreign key column in T_{fkey} points to a primary key column in T_{pkey}.

Each table T comprises a set of rows $T = \{v_1, \ldots, v_{n_T}\}$, where each row $v \in T$ includes three components: (1) Primary key p_v uniquely identifies the row v within its table. (2) Foreign keys $\mathcal{K}_v \subseteq \{p_{v'} : v' \in T' \text{ and } (T, T') \in \mathcal{L}\}$ establish connections to rows v' in other tables T'. (3) Attributes x_v hold the informational content of the row, such as textual data, which are encoded using `Txn-Bert` to generate embeddings.

We also define the relation type $\mathcal{R} = \mathcal{L} \cup \mathcal{L}^{-1}$, where $\mathcal{L}^{-1} = \{(T_{\text{pkey}}, T_{\text{fkey}}) | (T_{\text{fkey}}, T_{\text{pkey}}) \in \mathcal{L}\}$, representing the inverse of the primary-foreign key links. The heterogeneous graph is formalized as $G = (\mathcal{V}, \mathcal{E}, \phi, \psi)$:

1. $\mathcal{V}$: Node set, representing rows across all tables.
2. $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$: Edge set, representing relationships based on primary-foreign key mappings.
3. $\phi : \mathcal{V} \to \mathcal{T}$: Node type mapping function, assigning each node to its corresponding node type (table).
4. $\psi : \mathcal{E} \to \mathcal{R}$: Edge type mapping function, linking each edge to its relation type.

We then map the elements of the relational database $(\mathcal{T}, \mathcal{L})$ to the elements of the heterogeneous graph $G = (\mathcal{V}, \mathcal{E}, \phi, \psi)$. We first define the node set in the converted graph as the union of all rows in all tables $\mathcal{V} = \bigcup_{T \in \mathcal{T}} T$. Its edge set is then defined as

$$\mathcal{E} = \{(v_1, v_2) \in \mathcal{V} \times \mathcal{V} \mid p_{v_2} \in \mathcal{K}_{v_1} \text{ or } p_{v_1} \in \mathcal{K}_{v_v}\}. \tag{7.1}$$

That is, the edge set is the row pairs that arise from the primary-foreign key relationships in the database. Therefore, the type mapping function can be defined

as $\phi(v) = T$ for all $v \in T$ and $\psi(v_1, v_2) = (\phi(v_1), \phi(v_2)) \in \mathcal{R}$ if $(v_1, v_2) \in \mathcal{E}$. The attributes x_v hold the node attributes for each node v.

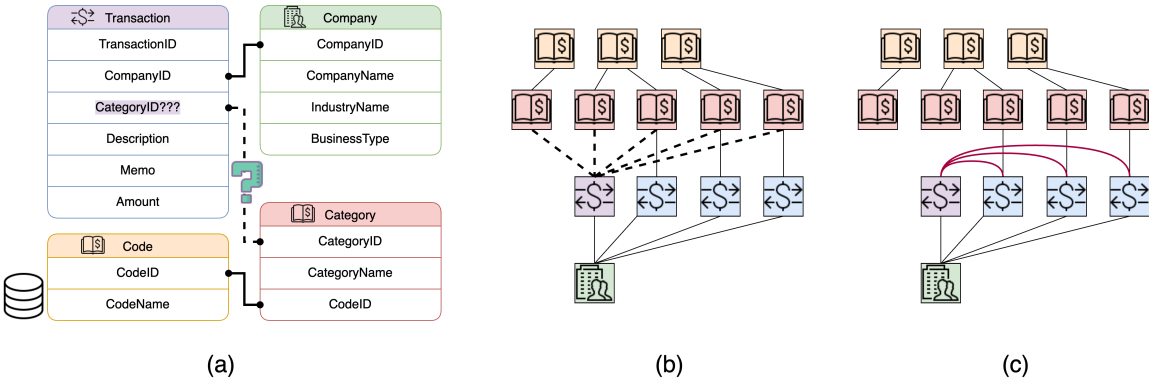

Figure 7.6: Transformation of a relational database into a heterogeneous graph for transaction categorization. (a) A new transaction enters the system without a foreign key connection to the *Category* table. (b) The heterogeneous graph is built from the relational database, where transaction categorization is formulated as a link prediction task. (c) Two-hop connections for transaction nodes mitigate over-squashing and improve model expressiveness.

7.4.2.1 Transform to a link prediction task

In the heterogeneous graph we've constructed, historical transaction nodes $v \in T_{\text{transaction}}$, that have already been categorized form links with corresponding *Category* nodes, such that $p_{v'} \in \mathcal{K}_v$ where $v' \in T_{Category}$. However, new transactions that have not yet been categorized enter the graph without any existing links to *Category* nodes ($p_{v'} \notin \mathcal{K}_v$ where $v' \in T_{Category}$). This scenario redefines the transaction categorization task. Instead of assigning a *Category* to each new transaction, our task shifts to predicting which *Category* node in the graph should be linked to the new transaction node. Essentially, the categorization challenge is transformed into a

link prediction task as shown in Figure 7.6 (a), where the objective is to determine the most appropriate connections for uncategorized transaction nodes based on the graph's existing structure and the attributes of its nodes.

7.4.3 Model the heterogeneous graph

This section details how `Rel-Cat` models the heterogeneous graph G. We employ a message-passing Graph Neural Network (GNN) [57], specifically a variant of GraphSAGE [59], to encode node representations. Given the graph's diverse node and edge types, we adapt our approach to model message-passing along different edge types separately. Each message type is initially processed through a homogeneous GNN. Messages from various edge types are then combined using a second-level aggregation function to get a comprehensive node representation:

$$\begin{aligned}\mathbf{h}_v^{(i+1)} =& t_{\phi(v)}\Big(\mathbf{h}_v^{(i)}, \mathrm{AGG}_{\mathrm{Heter}}(\{\mathrm{AGG}_{\mathrm{Homo}}(\{g_R(\mathbf{h}_w^{(i)}) \mid \\ & w \in \mathcal{N}_R(v)\}) \,\Big|\, \forall R = (T, \phi(v)) \in \mathcal{R}\})\Big),\end{aligned} \tag{7.2}$$

where $\mathcal{N}_R(v) = \{w \in \mathcal{V} \mid (w, v) \in \mathcal{E} \text{ and } \psi(w, v) = R\}$ is the neighborhood of node v under the specific edge type R. Here, $\mathrm{AGG}_{\mathrm{Homo}}$ employs a mean operator to normalize message contributions within the same type, while $\mathrm{AGG}_{\mathrm{Heter}}$ uses an attention mechanism [147] to dynamically weight the importance of messages from different types, enhancing the model's ability to prioritize relevant incoming messages.

7.4.3.1 Two-hop connections for transaction nodes

Our initial node representation learning framework in `Rel-Cat`, as outlined in Equation 7.2, aggregates messages from immediate neighbors within the graph G. While effective, this approach can introduce issues inherent in GNNs such as over-squashing [7] and limited expressiveness [167], which we address through graph

data augmentation.

Graph data augmentation. When GNNs learn the node representation of the target transaction in Figure 7.6 (b), they have to propagate at least two rounds of message passing to receive the information from the historical transactions of the same company. To enhance node representation, especially for the target transaction node, we introduce an additional type of edge within transaction nodes, effectively making historical transactions direct neighbors of the target transaction. This adjustment allows the target transaction to aggregate messages from historical transactions in just one message-passing iteration. The augmented graph with these new connections is shown in Figure 7.6 (c). For this new edge type (transaction to transaction), we utilize the GATv2 [16] as the aggregation operator (AGG_{Homo}). While applying GATv2 to all edge types can lead to GPU memory issues, using it for this specific edge type does not significantly increase memory requirements.

Specifically, we introduce an extra set of edges $\mathcal{E}_{\text{aug}}$ into the original graph:

$$\mathcal{E}_{\text{aug}} = \{(v_1, v_2) \in \mathcal{V} \times \mathcal{V} \mid v_1, v_2 \in T_{\text{transaction}}, \exists v_c \in T_{\text{company}}, p_c \in \mathcal{K}_{v_1} \cap \mathcal{K}_{v_2}\}.$$

The edge set $\mathcal{E}_{\text{aug}}$ is the set of transaction pairs if they are from the same company. Next, we discuss how this modification can mitigate the two issues.

Addressing over-squashing. GNNs learn the node representation iteratively from the local neighborhood. However, due to this recurrent learning paradigm, GNNs often face a challenge where the information from a growing receptive field is compressed into a fixed-length vector, potentially losing valuable information. This phenomenon is prevalent in the transaction graph G. For the target transaction in Figure 7.6 (b), the message from its historical transactions is equally compressed into one fixed-length vector. Since this message is not conditioned on the target

transaction, it cannot adjust itself so that the signals more important to the target transaction are preserved. By rewiring the graph and applying a weighted message aggregator like GATv2, it can effectively enable the target transaction's incoming message to keep the most relevant information [7], enhancing the effectiveness of the prediction.

Enhancing expressiveness. GNNs essentially simulate the Weisfeiler-Lehman graph isomorphism algorithm [167]. This limits their expressiveness in terms of distinguishing non-isomorphic graphs. By directly connecting two-hop neighbors (historical transactions) as immediate neighbors to the target transaction in graph G, our model approximates a K-hop GNN approach [48]. This augmentation not only improves expressiveness but does so without significantly increasing computational costs, thus maintains a balance between accuracy and efficiency.

7.4.4 Training objective

The transformation of the relational database into the heterogeneous graph G redefines our task as a link prediction challenge. In essence, link prediction in this context is a ranking problem where the model is expected to rank the correct node pair (the ground truth link) higher than other node pairs (non-connected links). Specifically, for a transaction node v_i and its corresponding *Category* node v_j in graph G, the score of (v_i, v_j) should be higher than that of any other *Category* node pair (v_i, v_k), where v_k represents any *Category* node not connected to v_i. We utilize the inner product as the scoring function between transaction and *Category* nodes, employing AUCLoss [157], a surrogate for AUC, as the training objective of `Rel-Cat`:

$$\mathcal{L} = \sum_{(v_i,v_j)\in\mathcal{E}^+,(v_i,v_k)\in\mathcal{E}^-} \left(1 - (\mathbf{h}_i * \mathbf{h}_j) + (\mathbf{h}_i * \mathbf{h}_k)\right)^2, \tag{7.3}$$

where $\mathbf{h}_v$ denotes the final representation of either a transaction or *Category* node from Equation 7.2. The set of positive node pairs $\mathcal{E}^+$ is chosen as:

$$\mathcal{E}^+ \subseteq \{(v_i, v_j) \in T_{\text{transaction}} \times T_{Category} \mid p_{v_j} \in \mathcal{K}_{v_i}\}. \tag{7.4}$$

The set of negative node pairs, $\mathcal{E}^-$, includes non-connected links, defined as:

$$\mathcal{E}^- \subseteq \{(v_i, v_k) \in T_{\text{transaction}} \times T_{Category} \mid p_{v_j} \notin \mathcal{K}_{v_i}\}. \tag{7.5}$$

7.4.4.1 Weighted negative sampling

The set of negative node pairs, $\mathcal{E}^-$, consists of non-connected links. Due to the impracticality of enumerating all potential negative pairs, we employ a negative sampling strategy. For a transaction node v_i and its corresponding positive *Category* node v_j, simple uniform sampling from $T_{Category} \setminus \{v_j\}$ would be suboptimal due to the long-tail distribution of *Category*. This could prevent effective learning if those *Category* nodes v_k, which rarely appear as positives, are sampled as negatives. To address this, we employ a weighted multinomial distribution for negative sampling, where weights are assigned proportional to the normalized frequency of *Category* appearances, enhancing the relevance and challenge of the sampled negatives.

7.4.4.2 Distribution shift mitigation

During training, the existing edge connections between transaction and *Category* nodes in $\mathcal{E}^+$ can potentially create a distribution shift [37]. These connections can be inadvertently learned as shortcuts by the GNN, which are not available during testing, thereby affecting model performance. To mitigate this, we adopt a strategy similar to the FakeEdge method [37], but implement in batch mode. Specifically, in each training epoch, we randomly select approximately 5% of all positive node pairs

from $\mathcal{E}^+$ to serve as our positive set $\mathcal{E}^+$. Then, we mask the edges in the graph if the node pairs are from the positive set:

$$\mathcal{E} := \mathcal{E} \setminus \mathcal{E}^+. \tag{7.6}$$

This separation creates two distinct edge groups within the graph: one that provides the training signals without any direct links ($\mathcal{E}^+$), and another that maintains the graph structure for message passing ($\mathcal{E} \setminus \mathcal{E}^+$). This ensures that during training, the connections used for supervision do not give away the actual links, thus preventing the model from leveraging these as shortcuts and better simulating the conditions it will encounter during testing.

7.4.5 Scalability and practical designs

In this section, we detail several scalability improvements to `Rel-Cat`, designed to effectively manage the extensive volume of transaction data encountered in real-world applications. We discuss practical designs to reduce neighborhood sizes for transaction and *Category* nodes during node representation encoding and introduce a rule-based early exit method, Top K Nearest Neighbor, to efficiently manage the workload on GNNs.

7.4.5.1 Reducing neighborhood size

Given that GNNs encode node representations by aggregating features from local neighborhoods, large neighborhoods can pose significant scalability challenges, particularly in terms of memory consumption. To address this, we implement a node-wise neighbor sampling strategy. Different from existing methods [53, 59, 172], we further take node types into consideration to optimize the receptive field of the GNNs. We employ distinct strategies for transaction nodes and *Category* nodes:

Similarity-based neighbor sampling for transaction nodes. For transaction nodes, $\{v \mid v \in T_{\text{transaction}}\}$, the local neighborhood typically includes all historical transactions associated with the company, which can be extensive. For example, there are companies owning over 2000 transactions within just one year. To manage this, we utilize similarity-based neighbor sampling, which reduces neighborhood size while preserving relevant information.

We compute the cosine similarity between the text embeddings of the target transaction node and its historical counterparts. Historical transactions are then sampled based on their similarity scores, prioritizing those most relevant to the target. This approach not only constrains computational overhead but also ensures that the most informative connections are maintained in `Rel-Cat`'s neighborhood. Techniques like Faiss [42] can be employed to enhance the efficiency of these computations.

Edge direction dropping for *Category* nodes. *Category* nodes, $\{v \mid v \in T_{Category}\}$, often have neighborhoods that include a huge set of transaction nodes linked to them. The popularity of some *Category* can lead to extremely large neighborhood sizes, which are not only impractical to process, but also ineffective to model.

To mitigate this, we discard all incoming edge connections from transaction to *Category* nodes. This adjustment significantly reduces the computational load by limiting the receptive field to exclude transaction nodes, while outgoing connections from *Category* to transaction nodes are kept. This approach not only simplifies the computation required to encode *Category* node but also ensures consistency in *Category* node representation during inference, as the computational graph remains unchanged regardless of new transactions being added.

7.4.5.2 Top K Nearest Neighbor

Despite the sophisticated capabilities of `Rel-Cat`, equipped with GNNs to perform predictions based on graph structure G, real-world scenarios often present varying degrees of prediction difficulty [119]. Many transactions imported into QuickBooks by users are very similar or even identical to past entries due to routine business activities. In such cases, users frequently reuse the same *Category* as in previous transactions. Consequently, a significant portion of transactions could be accurately categorized without necessitating full GNN processing. To capitalize on this, we introduce a streamlined and effective early exit method named Top K Nearest Neighbor (`TopK NN`).

Upon the arrival of a new transaction, while we still prepare the graph G for subsequent GNN processing, we first assess if sufficiently similar historical transactions might already provide reliable predictions. Utilizing the similarity scores computed during the similarity-based neighbor sampling of transaction nodes, we identify a subset of historical transactions that exhibit a similarity score exceeding a cutoff, set at 0.8 for our experiments.

From this subset, we directly derive *Category* from the top K most similar transactions. Given that our categorization task aims to predict the 5 most likely categories, `Rel-Cat` will output these labels directly if 5 distinct *Category* are available from this subset. If fewer than 5 categories are available, the graph G is then processed by the GNNs to generate the remaining predictions.

7.5 Experiments

7.5.1 Experimental setup

7.5.1.1 Dataset

To evaluate the performance of `Rel-Cat` comprehensively, we curated a dataset from active QuickBooks users as of November 2023. We randomly selected $7,500$ companies and used their two most recent transactions with labeled *Category* post-November 2023 as our test set, resulting in a total of $15,000$ transactions. The remaining companies were included in our training set. This approach ensures our evaluation set reflects a wide range of user preferences and patterns.

7.5.1.2 Experimental settings

We benchmark the performance of `Rel-Cat` against the current production models in QuickBooks, namely `Shorthair` and `Lynx`. We assess the models using the metrics of Top 1, Top 2, and Top 5 accuracy, which measure whether the correct label is among the Top k predictions of the model, ranked by the *Category* scores. We conduct evaluations under two distinct settings:

Zero Shot: In this setting, categorization is based solely on the information from the new transaction itself, without any contextual data from the owning company or its historical transactions. Both `Shorthair` and `Txn-Bert` are evaluated under this setting.

Few Shot: This setting incorporates not only the data from the new transaction but also contextual information from the owning company and its historical data. `Lynx`, `TopK NN`, and `Rel-Cat` are assessed under this framework.

TABLE 7.1

TRANSACTION CATEGORIZATION EVALUATED BY ACCURACY UNDER ZERO SHOT AND FEW SHOT SETTINGS.

Methods	**Top 1**	**Top 2**	**Top 5**
Zero Shot			
`Shorthair`	36.07	-	-
`Txn-Bert` (6 layers)	43.83	57.96	74.12
`Txn-Bert` (12 layers)	**45.52**	**59.47**	**75.46**
Few Shot			
`Lynx`	62.49	-	-
`TopK NN`	65.80	73.63	78.55
`Rel-Cat` (GNNs only)	63.04	74.60	85.19
`Rel-Cat`	**69.38**	**79.34**	**87.74**
Ablation Study			
`Rel-Cat` (GNNs only)	63.04	74.60	85.19
w/o `Txn-Bert`	55.46	64.02	73.16
w/o two-hop connections	47.07	61.16	76.58
w/o similarity sampling	56.39	67.89	80.63
w/o GATv2 on transactions	52.07	66.11	81.05
w/o weighted negative sampling	35.02	50.48	68.93
w/o distribution shift mitigations	59.40	71.83	84.12

7.5.2 Results

The experimental results are presented in Table 7.1. In the Zero Shot setting, `Txn-Bert` outperforms the production `Shorthair` model significantly. The `Txn-Bert` model with 6 layers achieves a Top 1 accuracy boost of 7.76% and a Top 5 accuracy of 74.12%, indicating that our trained-from-scratch text encoder effectively captures the semantics of transaction descriptions and maps them accurately to the corresponding *Category*. The 12-layer `Txn-Bert` model shows only marginal improvement over the 6-layer model, suggesting that a lightweight language model pretrained from scratch is sufficient for encoding transaction data.

In the Few Shot setting, `Rel-Cat` demonstrates substantial performance advantages. It achieves a Top 1 accuracy of 68.67% and a Top 5 accuracy of 88.04%, significantly outperforming the `Lynx` production model. Additionally, the `TopK NN` method, which uses text embeddings from `Txn-Bert`, achieves a Top 1 accuracy of 65.80%, surpassing the performance of `Rel-Cat` with GNNs alone. This result highlights the effectiveness of `TopK NN` in identifying recurring transactions in a user's history, thereby enhancing Top 1 accuracy of `Rel-Cat`.

However, when the transaction categorization must be inferred beyond the user's most similar historical transactions, the GNN component of `Rel-Cat` exhibits superior generalizability, achieving over 85% in Top 5 accuracy. This demonstrates that while `TopK NN` is highly effective for repeated transactions, the GNN module of `Rel-Cat` provides a broader and more accurate categorization capability for diverse transaction scenarios.

7.5.3 Ablation Studies

In this section, we delve deeper into the validation of the effectiveness of various design choices in `Rel-Cat`.

In Table 7.1, we report ablation studies to validate if the proposed components in `Rel-Cat` can enhance the predictive power of transaction categorization. For the ablation studies, we only include the GNNs module for `Rel-Cat`.

w/o ***Txn-Bert*** *(Sec 7.3):* Replacing `Txn-Bert` with off-the-shelf Sentence-Bert [125], we observe a decrease in performance, underscoring the necessity of a trained-from-scratch text encoder tailored to transaction data. We further note the steep drop in performance for the unseen subset.

w/o two-hop connections (Sec 7.4.3.1): This ablation study emphasizes the critical role of explicit graph augmentation. It demonstrates that directly connecting the target and historical transactions as neighbors significantly enhances performance.

This finding underscores that merely converting a relational database to a heterogeneous graph without strategic modifications is inadequate for maximizing the model's effectiveness. While this setting has the best performance for the unseen subset, it comes at the expense of significant decline in overall performance.

w/o similarity sampling (Sec 7.4.5.1): Testing the impact of neighbor sampling based on semantic similarity, we find that sampling similar historical transactions in GNN's computation graphs allows `Rel-Cat` to leverage relevant information effectively for accurate predictions. We note that this setting also suffers from lack of generalizability.

w/o GATv2 on transactions (Sec 7.4.3.1). Examining the necessity of GATv2 for transaction-to-transaction message passing, the results indicate that such a GNN structure enhances `Rel-Cat`'s performance by enabling conditioned message passing.

w/o weighted negative sampling (Sec 7.4.4.1). This study evaluates the importance of weighted negative sampling in training `Rel-Cat`. Findings suggest that this approach is crucial for achieving optimal performance and addressing the challenges posed by the skewed distribution of *Category* labels.

w/o distribution shift mitigations 7.4.4.2. Assessing our approach to mitigating distribution shift shows that this is a significant issue in link prediction tasks, which can be effectively managed with strategic consideration of message-passing and supervision signal edges.

7.5.4 Seen vs Unseen Category in a Company's history

In Table 7.2, we not only report our overall accuracy, but further break down the performance of `Rel-Cat` into Historical Seen and Historical Unseen scenarios. For Historical Seen, we choose the test samples such that their ground-truth *Category*

TABLE 7.2

PERFORMANCE BREAKDOWN IN DIFFERENT SCENARIOS.

Methods	**Acc**	**HS**	**HU**
`TopK NN`	65.80	**79.72**	0.16
`Rel-Cat` (GNNs only)	63.04	71.15	**24.95**
`Rel-Cat`	**69.38**	79.23	23.09
Methods (GNNs only)	**Acc**	**HS**	**HU**
`Rel-Cat` (GNNs only)	63.04	71.15	24.95
w/o `Txn-Bert`	55.46	66.24	4.83
w/o two-hop connections	47.07	51.36	26.93
w/o similarity sampling	56.39	65.26	14.73

Acc is the overall Top 1 accuracy, **HS** is the accuracy on Historical Seen subset, and **HU** is the accuracy on Historical Unseen subset.

is present within the company's own history as context. For Historical Unseen, we choose the samples whose *Category* are present in the overall dataset but unseen to that company's history. We note that `TopK NN`, while having the best performance at repeated labels, has no predictive power for unseen labels. For this subset, `Rel-Cat`'s GNN is able to detect the labels with nearly 25% accuracy, highlighting its generalizable ability. `Rel-Cat` has the best overall performance, balancing between both the subsets, thereby demonstrating the need for GNN and `TopK NN` hybrid model. Furthermore, analyzing these trends over the ablation studies, we note that each design choice adds to the overall accuracy, augmenting performance for either or both the seen and unseen subsets.

TABLE 7.3

INFERENCE TIMES FOR 1,000 TRANSACTIONS.

Walltime (ms)	**CPU**	**GPU**
`Txn-Bert`	1400	67
`Rel-Cat`	6808	843
- `TopK NN`	333	-
- `Rel-Cat` (GNNs only)	6614	324
Total	8208	910

7.5.4.1 Time complexity

We have assessed the processing time for various components within `Rel-Cat`, with the results detailed in Table 7.3. We measure the processing time for 1,000 transactions in milliseconds (ms). The entire pipeline of `Rel-Cat` is designed to operate effectively on both CPU and GPU environments, catering to different production system requirements. On a CPU, the lightweight text encoder `Txn-Bert` processes 1,000 transactions in just 1400 milliseconds. To deliver the top 5 predictions, `Rel-Cat` requires approximately 8 seconds for 1,000 transactions. In contrast, utilizing a GPU significantly reduces the processing time to under 1 second for outputting the top 5 predictions. This efficiency demonstrates `Rel-Cat`'s capability to scale and meet the demands of real-world transaction volumes effectively.

7.6 Conclusion

In this study, we introduce `Rel-Cat`, a unified model designed for transaction categorization within QuickBooks. Recognizing the unique linguistic characteristics of transaction data, we start by developing a trained-from-scratch `Txn-Bert` text encoder, specifically trained to grasp the semantic nuances of transaction descriptions.

Subsequently, we employ a GNN-based approach to model the intricate relationships among the tables in a relational database, redefining transaction categorization as a link prediction task. To address the challenges posed by the high volume of transactions and the specific demands of the categorization task, we integrate several innovative components that significantly boost `Rel-Cat`'s predictive capabilities. Our experimental results demonstrate that `Rel-Cat` not only surpasses previous production models in terms of performance but also offers remarkable scalability and efficiency, making it well-suited for handling large-scale transaction data in real-world settings.

CHAPTER 8

CONCLUSION AND FUTURE DIRECTIONS

In this thesis, we have addressed key challenges that currently limit the broader adoption and effectiveness of Graph Neural Networks (GNNs) in real-world applications. We have specifically focused on improving GNNs from three critical aspects: robustness, efficiency, and adaptability.

In Part I, we examined the robustness of GNNs, highlighting two fundamental limitations: dataset shift and data noise. The structural mismatch between training and testing phases, known as dataset shift, significantly impacts the reliability of GNN-based models in link prediction (LP). To mitigate this, we developed FakeEdge, a model-agnostic method designed to maintain consistent link representations by correcting the structural gaps between training and testing graphs. Furthermore, acknowledging that noisy and incomplete graph data directly compromise GNN performance, we proposed COmplete and REduce (CORE), a novel data augmentation framework that systematically recovers missing edges and removes harmful noise, thus enhancing model robustness from a data-driven perspective.

In addressing the efficiency of GNNs in Part II, we highlighted two core computational issues that restrict their efficiency: the complexity involved in capturing essential graph substructures, and the heavy computational burden associated with iterative training processes. To overcome these challenges, we proposed Message Passing Link Predictor (MPLP), a method that efficiently estimates critical link-level substructures while maintaining computational complexity at the node level. Additionally, we introduced a training-free paradigm, TrainlessGNN, specifically for

semi-supervised node classification on text-attributed graphs. By directly approximating optimal parameters through node attributes and graph structures, this approach eliminates the need for computationally expensive iterative training, significantly enhancing the practicality and scalability of GNN models.

We further investigated GNN adaptability in scenarios requiring models to generalize beyond individual graphs and even beyond inherently graph-based data. To address the limited transferability of traditional GNNs, we developed the Universal Link Predictor (UniLP), a framework capable of leveraging prior knowledge from previously encountered graphs and adapting dynamically to unseen target graphs during inference. UniLP achieves this adaptability by utilizing an attention-based mechanism inspired by In-context Learning. Beyond traditional graph-structured scenarios, we also expanded GNN applicability into the domain of relational databases—an area traditionally overlooked by graph-based deep learning approaches. By transforming relational databases into heterogeneous graphs, we demonstrated that GNNs could effectively handle real-world business tasks without explicit feature engineering, significantly enhancing their adaptability and practical relevance.

Collectively, the methods developed in this thesis advance the field of graph machine learning, enhancing GNN robustness, efficiency, and adaptability. This thesis not only empowers GNNs with greater ability to tackle complex problems in the real world, but also opens several avenues for further advancements and exploration in AI throughout this dissertation. Moving forward, I intend to pursue research in several key directions:

Knowledge transfer across graphs. Recent years have shown impressive success in pretraining models on large-scale datasets [18, 83]. Particularly within natural language processing (NLP), large language models trained on extensive internet-scale data have achieved remarkable performance across diverse downstream

tasks [32, 71, 115, 146]. The emergence of scaling laws [77] suggests that a similar potential for improvement could exist across other data modalities, including graphs. Thus, we expect that the modeling capacity of graph data could be significantly enhanced through extensive pretraining. Furthermore, knowledge derived from previously encountered graphs might be beneficial to new target graphs. We took an initial step toward investigating this possibility in [41], focusing specifically on knowledge transfer for link prediction tasks. However, several critical questions remain unresolved regarding the generalizability of graph machine learning. First, it remains unclear under which conditions transferring knowledge from one graph to another genuinely benefits downstream tasks; specifically, the extent to which data distributions differ across various graphs is still debatable. Second, graph machine learning tasks inherently span multiple levels (node-level, link-level, subgraph-level, and graph-level), and currently there is no unified approach to handling these different levels simultaneously. This limitation necessitates practical designs of graph foundation models [98] to have specialized prediction heads for each task level. Moving forward, I plan to deepen the investigation into cross-graph knowledge transfer, clearly identifying conditions under which transfer is beneficial. Additionally, I aim to explore methods for unifying tasks across different levels of graph machine learning into a single coherent framework, thus paving the way toward more versatile and generalizable graph models.

Practical applications of GNNs beyond graph-structured data. GNNs have demonstrated remarkable potential when applied to graph-structured data. The majority of existing studies have concentrated on established graph benchmarks, such as 2D molecular graphs [44], social networks [177], citation networks [169], biological networks [62], and knowledge graphs [195]. However, these benchmarks are naturally represented in graph form, limiting the broader applicability of GNNs. To fully

harness the relational modeling capabilities of GNNs, future studies should investigate scenarios where data is not inherently structured as graphs. In Chapter 7, we successfully demonstrated the application of GNNs to relational databases by solving a practical transaction categorization task. Motivated by this success, I intend to further explore the application of GNNs to other domains lacking explicit graph structures, including relational databases in diverse contexts [126], computer systems [121], and combinatorial optimization problems [21]. Such investigations will significantly expand the scope and real-world impact of GNN methodologies.

APPENDIX A

FAKEEDGE: ALLEVIATE DATASET SHIFT IN LINK PREDICTION

A.1 Proof of Theorem 1

We restate the Theorem 1: `GNN` cannot learn the subgraph feature $\mathbf{h}$ to be Edge Invariant.

Proof. Recall that the computation of subgraph feature $\mathbf{h}$ involves steps such as:

1. **Subgraph Extraction**: Extract the subgraph $\mathcal{G}_{i,j}^r$ around the focal node pair $\{i, j\}$;
2. **Node Representation Learning**: $\boldsymbol{Z} = \texttt{GNN}(\mathcal{G}_{i,j}^r)$, where $\boldsymbol{Z} \in \mathbb{R}^{|V_{i,j}^r| \times F_{\text{hidden}}}$ is the node embedding matrix learned by the GNN encoder;
3. **Pooling**: $\mathbf{h} = \texttt{Pooling}(\boldsymbol{Z}; \mathcal{G}_{i,j}^r)$, where $\mathbf{h} \in \mathbb{R}^{F_{\text{pooled}}}$ is the latent feature of the subgraph $\mathcal{G}_{i,j}^r$;

Here, `GNN` is Message Passing Neural Network [57]. Given a subgraph $\mathcal{G} = (V, E, \boldsymbol{x}^V, \boldsymbol{x}^E)$, `GNN` with T layers applies following rules to update the representation of node $i \in V$:

$$h_i^{(t+1)} = U_t(h_i^{(t)}, \sum_{w \in \mathcal{N}(i)} M_t(h_i^{(t)}, h_w^{(t)}, \boldsymbol{x}_{i,w}^E)), \tag{A.1}$$

where $\mathcal{N}(i)$ is the neighborhood of node i in $\mathcal{G}$, M_t is the message passing function at layer t and U_t is the node update function at layer t. The hidden states at the first layer are set as $h_i^{(0)} = \boldsymbol{x}_i^V$. The hidden states at the last layer are the outputs $\boldsymbol{Z}_i = h_i^{(T)}$.

Given any subgraph $\mathcal{G}^r_{i,j} = (V^r_{i,j}, E^r_{i,j}, \boldsymbol{x}^V_{V^r_{i,j}}, \boldsymbol{x}^E_{E^r_{i,j}})$ with the edge present at the focal node pair $(i,j) \in E^r_{i,j}$, we construct another isomorphic subgraph $\mathcal{G}^r_{\bar{i},\bar{j}} = (V^r_{\bar{i},\bar{j}}, E^r_{\bar{i},\bar{j}}, \boldsymbol{x}^V_{V^r_{\bar{i},\bar{j}}}, \boldsymbol{x}^E_{E^r_{\bar{i},\bar{j}}})$, but remove the edge $(\bar{i},\bar{j})$ from the edge set $E^r_{\bar{i},\bar{j}}$ of the subgraph. $\mathcal{G}^r_{\bar{i},\bar{j}}$ can be seen as the counterpart of $\mathcal{G}^r_{i,j}$ in the testing set.

Thus, for the first iteration of node updates $t = 1$:

$$h_i^{(1)} = U_t(h_i^{(0)}, \sum_{w \in \mathcal{N}(i)} M_t(h_i^{(0)}, h_w^{(0)}, \boldsymbol{x}^E_{i,w})), \tag{A.2}$$

$$h_{\bar{i}}^{(1)} = U_t(h_{\bar{i}}^{(0)}, \sum_{w \in \mathcal{N}(\bar{i})} M_t(h_{\bar{i}}^{(0)}, h_w^{(0)}, \boldsymbol{x}^E_{\bar{i},w})), \tag{A.3}$$

Note that $\mathcal{N}(\bar{i}) \cup \{j\} = \mathcal{N}(i)$. We have:

$$h_i^{(1)} = U_t(h_i^{(0)}, \sum_{w \in \mathcal{N}(i) \setminus \{j\}} M_t(h_i^{(0)}, h_w^{(0)}, \boldsymbol{x}^E_{i,w}) + M_t(h_i^{(0)}, h_j^{(0)}, \boldsymbol{x}^E_{i,j})) \tag{A.4}$$

$$= U_t(h_{\bar{i}}^{(0)}, \sum_{w \in \mathcal{N}(\bar{i})} M_t(h_{\bar{i}}^{(0)}, h_w^{(0)}, \boldsymbol{x}^E_{\bar{i},w}) + M_t(h_{\bar{i}}^{(0)}, h_{\bar{j}}^{(0)}, \boldsymbol{x}^E_{\bar{i},\bar{j}})), \tag{A.5}$$

As U_t is injective, $p(h_i^{(1)}, \mathrm{y} = 1 | \mathrm{e} = 1) \neq p(h_{\bar{i}}^{(1)}, \mathrm{y} = 1) = p(h_i^{(1)}, \mathrm{y} = 1 | \mathrm{e} = 0)$. Similarly, we can conclude that $p(h_i^{(T)}, \mathrm{y} = 1 | \mathrm{e} = 1) \neq p(h_i^{(T)}, \mathrm{y} = 1 | \mathrm{e} = 0)$.

As we use the last iteration of node updates $h_i^{(T)}$ as the final node representation $\boldsymbol{Z}$, we have $p(\boldsymbol{Z}, \mathrm{y} | \mathrm{e} = 1) \neq p(\boldsymbol{Z}, \mathrm{y} | \mathrm{e} = 0)$, which leads to $p(\mathrm{h}, \mathrm{y} | \mathrm{e} = 1) \neq p(\mathrm{h}, \mathrm{y} | \mathrm{e} = 0)$ and concludes the proof. □

A.2 Proof of Theorem 2

We restate the Theorem 2: Given $p(\mathbf{h}, \mathrm{y} | \mathrm{e}, \mathrm{c}) = p(\mathbf{h}, \mathrm{y} | \mathrm{e})$, there is no **Dataset Shift** in the link prediction if the subgraph embedding is **Edge Invariant**. That is, $p(\mathbf{h}, \mathrm{y} | \mathrm{e}) = p(\mathbf{h}, \mathrm{y}) \Longrightarrow p(\mathbf{h}, \mathrm{y} | \mathrm{c}) = p(\mathbf{h}, \mathrm{y})$.

Proof.

$$p(\mathbf{h} = \boldsymbol{h}, \mathrm{y} = y | \mathrm{c} = c) \tag{A.6}$$

$$= \mathbb{E}_{\mathrm{e}}[p(\mathbf{h} = \boldsymbol{h}, \mathrm{y} = y | \mathrm{c} = c, \mathrm{e}) p(\mathrm{e} | \mathrm{c} = c)] \tag{A.7}$$

$$= \mathbb{E}_{\mathrm{e}}[p(\mathbf{h} = \boldsymbol{h}, \mathrm{y} = y) p(\mathrm{e} | \mathrm{c} = c)] \tag{A.8}$$

$$= p(\mathbf{h} = \boldsymbol{h}, \mathrm{y} = y). \tag{A.9}$$

□

A.3 Details about the baseline methods

To verify the effectiveness of FakeEdge, we tend to introduce minimal modification to the baseline models and make them compatible with FakeEdge techniques. The baseline models in our experiments are mainly from the two streams of link prediction models. One is the GAE-like model, including GCN [82], SAGE [59], GIN [167] and PLNLP [157]. The other includes SEAL [178] and WalkPool [118]. GCN, SAGE and PLNLP learn the node representation and apply a score function on the focal node pair to represent the link. As GAE-like models are not implemented in the fashion of subgraph link prediction, the subgraph extraction step is necessary for them as preprocessing. We follow the code from the labeling trick [182], which implements the GAE models as the subgraph link prediction task. In particular, GIN concatenates the node embedding from different layers to learn the node representation and applies a subgraph-level readout to aggregate as the subgraph representation. As suggested by [90, 182], we always inject the distance information with the Double-Radius Node Labeling [178] (DRNL) to enhance the model performance of GAE-like models. In terms of the selection of hyperparameters, we use the same configuration as [182] on datasets Cora, Citeseer and Pubmed. As they do not have experiments on other 8 networks without attributes, we set the subgraph hop number as 2 and leave the

rest of them as default. For PLNLP, we also add a subgraph extraction step without modifying the core part of the pairwise learning strategy. We find that the performance of PLNLP under subgraph setting is very unstable on different train/test splits. In particular, the performance's standard deviation of PLNLP is over 10% on each experiment. Therefore, we also apply DRNL to stabilize the model.

SEAL and WalkPool have applied one of the FakeEdge techniques in their initial implementation. SEAL uses a *Edge Minus* strategy to remove all the edges at focal node pair as a preprocessing step, while WalkPool applies *Edge Plus* to always inject edges into the subgraph for node representation learning. Additionally, WalkPool has the walk-based pooling method operating on both the *Edge Plus* and *Edge Minus* graphs. This design is kept in our experiment. Thus, our FakeEdge technique only takes effect on the node representation step for WalkPool. From the results in Section 2.6.2, we can conclude that the dataset shift issue on the node representation solely would significantly impact the model performance. We also use the same hyperparameter settings as originally reported in their paper.

A.4 Benchmark dataset descriptions

The graph datasets with node attributes are three citation networks: **Cora** [102], **Citeseer** [56] and **Pubmed** [109]. Nodes represent publications and edges represent citation links. The graph datasets without node attributes are: (1) **USAir** [10]: a graph of US Air lines; (2) **NS** [110]: a collaboration network of network science researchers; (3) **PB** [3]: a graph of links between web pages on US political topic; (4) **Yeast** [150]: a protein-protein interaction network in yeast; (5) **C.ele** [158]: the neural network of Caenorhabditis elegans; (6) **Power** [158]: the network of the western USś electric grid; (7) **Router** [136]: the Internet connection at the router-level; (8) **E.coli** [180]: the reaction network of metabolites in Escherichia coli. The detailed statistics of the datasets can be found in Table A.1.

TABLE A.1

STATISTICS OF LINK PREDICTION DATASETS.

Dataset	#Nodes	#Edges	Avg. node deg.	Density	Attr. Dimension
Cora	2708	10556	3.90	0.2880%	1433
Citeseer	3327	9104	2.74	0.1645%	3703
Pubmed	19717	88648	4.50	0.0456%	500
USAir	332	4252	12.81	7.7385%	-
NS	1589	5484	3.45	0.4347%	-
PB	1222	33428	27.36	4.4808%	-
Yeast	2375	23386	9.85	0.8295%	-
C.ele	297	4296	14.46	9.7734%	-
Power	4941	13188	2.67	0.1081%	-
Router	5022	12516	2.49	0.0993%	-
E.coli	1805	29320	16.24	1.8009%	-

A.5 Results measured by Hits@20 and statistical significance of results

We adopt another widely used metrics in the link prediction task [62], Hits@20, to evaluate the model performance with and without FakeEdge. The results are shown in Table A.2. FakeEdge can boost all the models predictive power on different datasets.

Note that the AUC scores on several datasets are almost saturated in Table 2.1. To further verify the statistical significance of the improvement, a two-sided t-test is conducted with the null hypothesis that the augmented *Edge Att* and the *Original* representation learning would reach at the identical average scores. The p-values of different methods can be found in Table A.3. Recall that the p-value smaller than 0.05 is considered as statistically significant. GAE-like methods obtain significant improvement on almost all of the datasets, except GCN on C.ele. SEAL shows

TABLE A.2

COMPARISON WITH AND WITHOUT FAKEEDGE (HITS@20).

Models	FakeEdge	Cora	Citeseer	Pubmed	USAir	NS	PB	Yeast	C.ele	Power	Router	E.coli
GCN	*Original*	65.35±3.64	61.71±2.60	48.97±1.92	87.69±3.92	92.77±1.72	41.60±2.52	85.26±1.90	65.33±7.55	39.64±5.47	39.41±2.38	82.21±2.02
	Edge Plus	68.31±2.89	65.80±3.28	**55.70±3.07**	89.34±4.09	93.28±1.69	43.98±6.25	87.19±2.13	**66.68±5.25**	46.92±3.78	72.03±2.85	86.03±1.40
	Edge Minus	67.97±2.62	66.13±3.30	54.29±2.66	90.57±3.30	**93.61±1.68**	43.92±5.82	86.66±2.18	66.07±6.14	47.97±2.58	**72.34±2.58**	85.68±1.84
	Edge Mean	67.76±3.02	66.11±2.48	54.55±2.88	89.48±3.52	92.77±1.99	44.64±6.93	86.64±2.03	65.28±6.33	47.54±2.95	72.26±2.68	85.62±1.71
	Edge Att	**68.43±3.72**	**67.65±4.11**	55.55±2.70	**90.80±4.50**	92.88±2.27	**44.80±6.60**	**87.83±0.92**	65.93±11.06	**48.50±2.20**	70.96±2.85	**86.56±1.69**
SAGE	*Original*	61.67±3.68	61.10±1.54	45.29±3.99	89.20±2.80	91.93±2.74	39.51±4.44	84.11±1.47	58.55±7.17	42.97±5.34	30.02±2.75	75.30±2.77
	Edge Plus	68.58±2.77	65.47±3.58	**55.23±2.81**	92.59±3.71	93.83±2.54	**49.10±5.38**	**89.36±0.72**	69.72±6.02	**49.70±2.57**	**74.90±3.73**	**88.16±1.29**
	Edge Minus	66.26±2.54	62.97±3.50	53.43±3.52	91.32±3.42	93.54±1.96	48.72±4.90	88.27±1.00	**69.81±5.34**	47.63±1.87	56.67±7.20	87.89±1.59
	Edge Mean	66.74±2.71	65.96±2.62	55.21±2.84	91.51±3.49	93.25±2.88	48.89±6.14	89.30±0.72	69.21±7.17	47.54±3.52	73.89±3.50	88.05±1.62
	Edge Att	**68.80±2.65**	**66.62±3.67**	55.18±2.99	**92.92±3.11**	**94.09±1.60**	48.53±5.15	89.10±1.17	69.30±7.53	47.06±2.21	73.60±4.68	87.63±1.66
GIN	*Original*	55.71±4.38	51.71±4.31	40.14±3.98	86.08±3.14	90.51±3.45	38.79±5.32	79.57±1.74	54.95±5.91	41.56±1.47	55.47±4.37	77.37±2.84
	Edge Plus	**64.42±2.67**	63.56±2.92	49.75±4.50	88.68±4.10	**94.85±1.90**	**46.17±6.12**	87.58±2.22	64.49±6.52	**48.59±3.33**	70.67±3.58	84.13±2.12
	Edge Minus	63.17±2.96	63.65±4.63	**50.37±4.01**	89.81±1.80	94.53±2.09	45.93±6.09	**88.37±2.00**	**67.06±11.03**	47.56±1.88	**71.10±1.90**	83.23±2.62
	Edge Mean	61.46±4.64	**63.74±4.20**	46.97±6.49	**89.86±2.62**	93.98±2.88	43.48±7.74	88.16±2.11	66.73±6.79	47.66±2.91	71.09±2.68	82.48±1.99
	Edge Att	63.26±3.33	60.64±4.29	49.71±4.40	88.87±4.71	94.49±1.51	44.94±5.37	87.92±1.45	65.93±8.55	48.19±2.70	70.03±3.05	**84.38±2.54**
PLNLP	*Original*	58.77±2.59	57.21±3.91	40.03±3.46	88.87±2.75	93.76±1.65	38.90±4.38	81.17±3.54	66.36±5.65	43.52±6.47	34.61±11.29	65.68±1.56
	Edge Plus	66.79±2.77	**67.69±4.13**	44.44±14.29	95.19±1.60	95.84±1.09	45.18±4.87	88.04±2.42	71.21±8.04	**52.37±3.95**	**75.01±1.83**	84.73±1.70
	Edge Minus	67.40±3.53	62.84±2.88	47.80±11.11	94.10±2.42	95.22±1.60	**45.40±6.29**	87.94±1.64	69.91±6.80	52.19±4.23	68.24±4.01	83.59±1.56
	Edge Mean	**68.61±3.40**	64.81±3.57	**51.92±13.30**	95.24±2.09	**95.95±0.78**	45.37±5.07	88.08±2.30	**71.26±8.05**	51.97±3.41	74.42±2.33	84.78±1.82
	Edge Att	67.82±3.58	64.37±3.73	48.47±12.01	**95.38±2.02**	95.62±0.81	45.28±5.11	**88.57±1.80**	70.65±8.11	51.79±4.07	74.99±1.92	**85.10±1.88**
SEAL	*Original*	60.95±8.00	61.56±2.12	48.80±3.33	91.27±2.53	91.72±2.01	43.44±6.82	85.33±1.76	64.21±5.86	39.30±3.79	59.47±6.66	84.15±2.16
	Edge Plus	60.51±7.70	65.12±2.18	50.90±3.96	90.85±4.12	93.61±1.87	46.77±4.80	86.66±1.59	65.47±7.68	45.90±2.85	70.06±3.57	85.76±2.04
	Edge Minus	60.74±6.60	**65.14±2.93**	51.23±3.82	90.66±3.49	92.19±2.03	**47.21±4.73**	86.49±2.08	63.64±6.93	46.42±3.42	**70.43±4.40**	85.50±2.06
	Edge Mean	**62.94±5.78**	64.99±4.36	**51.83±3.66**	**91.84±2.93**	92.92±2.12	46.02±4.22	86.25±2.17	**65.93±6.87**	46.57±3.22	70.08±3.85	85.85±1.81
	Edge Att	62.03±4.95	63.52±4.39	48.42±5.69	91.42±3.31	**94.64±1.49**	44.73±5.29	**86.83±1.63**	65.93±4.74	**47.91±3.45**	67.46±3.49	**86.02±1.58**
WalkPool	*Original*	69.98±3.37	64.22±2.84	57.30±2.56	95.09±2.78	96.02±1.64	**47.74±5.81**	88.24±1.33	**78.55±5.83**	43.58±4.40	56.21±13.92	83.41±1.72
	Edge Plus	69.13±2.31	**64.51±2.25**	59.23±3.09	95.00±3.09	96.06±1.65	46.18±5.40	89.79±0.70	78.36±5.30	56.27±4.17	**77.65±2.83**	86.44±1.52
	Edge Minus	69.34±2.45	64.26±1.93	59.44±3.10	95.14±2.93	95.99±1.67	46.79±4.88	89.57±0.85	77.90±4.49	55.72±3.63	77.62±2.64	**87.24±0.77**
	Edge Mean	**70.27±2.96**	62.84±4.79	**59.85±3.84**	95.24±2.45	**96.17±1.63**	46.27±5.00	89.58±0.91	77.94±4.55	56.18±3.74	76.88±2.76	86.89±0.84
	Edge Att	69.60±4.11	64.35±3.64	59.63±3.28	**95.61±2.53**	96.06±1.62	46.77±5.36	**89.84±0.71**	77.94±4.89	**56.46±3.55**	76.90±2.82	87.02±1.64

TABLE A.3

p-VALUES BY COMPARING AUC SCORES WITH *ORIGINAL* AND *EDGE ATT*.

Models	Cora	Citeseer	Pubmed	USAir	NS	PB	Yeast	C.ele	Power	Router	E.coli
GCN	$\mathbf{3.50 \cdot 10^{-09}}$	$\mathbf{6.92 \cdot 10^{-12}}$	$\mathbf{1.52 \cdot 10^{-09}}$	$\mathbf{1.10 \cdot 10^{-05}}$	$\mathbf{9.89 \cdot 10^{-04}}$	$\mathbf{1.21 \cdot 10^{-09}}$	$\mathbf{4.95 \cdot 10^{-13}}$	$2.76 \cdot 10^{-01}$	$\mathbf{1.55 \cdot 10^{-05}}$	$\mathbf{2.62 \cdot 10^{-13}}$	$\mathbf{2.44 \cdot 10^{-14}}$
SAGE	$\mathbf{1.32 \cdot 10^{-08}}$	$\mathbf{2.04 \cdot 10^{-06}}$	$\mathbf{3.48 \cdot 10^{-14}}$	$\mathbf{2.78 \cdot 10^{-02}}$	$\mathbf{2.33 \cdot 10^{-02}}$	$\mathbf{4.13 \cdot 10^{-06}}$	$\mathbf{4.87 \cdot 10^{-08}}$	$\mathbf{1.23 \cdot 10^{-03}}$	$\mathbf{6.12 \cdot 10^{-10}}$	$\mathbf{4.40 \cdot 10^{-12}}$	$\mathbf{3.54 \cdot 10^{-13}}$
GIN	$\mathbf{4.86 \cdot 10^{-10}}$	$\mathbf{6.09 \cdot 10^{-11}}$	$\mathbf{1.46 \cdot 10^{-12}}$	$\mathbf{1.27 \cdot 10^{-03}}$	$\mathbf{1.29 \cdot 10^{-05}}$	$\mathbf{2.47 \cdot 10^{-10}}$	$\mathbf{5.34 \cdot 10^{-11}}$	$\mathbf{3.84 \cdot 10^{-04}}$	$\mathbf{5.10 \cdot 10^{-09}}$	$\mathbf{3.11 \cdot 10^{-16}}$	$\mathbf{3.04 \cdot 10^{-12}}$
PLNLP	$\mathbf{1.47 \cdot 10^{-10}}$	$\mathbf{5.30 \cdot 10^{-07}}$	$\mathbf{1.22 \cdot 10^{-06}}$	$\mathbf{1.66 \cdot 10^{-07}}$	$\mathbf{1.70 \cdot 10^{-02}}$	$\mathbf{3.40 \cdot 10^{-08}}$	$\mathbf{7.69 \cdot 10^{-06}}$	$\mathbf{2.46 \cdot 10^{-03}}$	$\mathbf{7.84 \cdot 10^{-06}}$	$\mathbf{2.68 \cdot 10^{-13}}$	$\mathbf{5.27 \cdot 10^{-11}}$
SEAL	$2.59 \cdot 10^{-01}$	$\mathbf{1.72 \cdot 10^{-02}}$	$\mathbf{6.45 \cdot 10^{-05}}$	$4.82 \cdot 10^{-01}$	$\mathbf{1.15 \cdot 10^{-02}}$	$5.20 \cdot 10^{-01}$	$\mathbf{5.91 \cdot 10^{-04}}$	$4.12 \cdot 10^{-01}$	$\mathbf{3.78 \cdot 10^{-06}}$	$\mathbf{3.91 \cdot 10^{-06}}$	$\mathbf{5.67 \cdot 10^{-04}}$
WalkPool	$9.52 \cdot 10^{-01}$	$4.96 \cdot 10^{-01}$	$\mathbf{2.83 \cdot 10^{-07}}$	$4.77 \cdot 10^{-01}$	$8.91 \cdot 10^{-01}$	$\mathbf{1.84 \cdot 10^{-05}}$	$\mathbf{1.07 \cdot 10^{-04}}$	$8.74 \cdot 10^{-01}$	$\mathbf{4.15 \cdot 10^{-07}}$	$\mathbf{5.89 \cdot 10^{-04}}$	$\mathbf{1.83 \cdot 10^{-10}}$

Significant differences are highlighted in bold.

significant improvement with *Edge Att* on 7 out of 11 datasets. For WalkPool, more than half of the datasets are significantly better.

A.6 FakeEdge with extremely sparse graphs

In real applications, the size of testing set often outnumbers the training set. When it happens to a link prediction task, the graph will become more sparse because of the huge number of unseen links. We are interested to see how FakeEdge can handle situations where the ratio of training set is low and there exists a lot of "true" links missing in the training graph.

We reset the train/test split as 20% for training, 30% for validation and 50% for testing and reevaluate the model performance. The results can be found in Table A.4. As shown in the table, FakeEdge can still consistently improve the model performance under such an extreme setting. It shows that the dataset shift for link prediction is a common issue and FakeEdge has the strength to alleviate it in various settings.

However, we still observe a significant performance drop when compared to the 85/5/10 evaluation setting. This degradation may be caused by a more fundamental dataset shift problem of link prediction: the nodes in a graph are not sampled inde-

pendently. Existing link prediction models often assume that the likelihood of forming a link relies on its local neighborhood. Nevertheless, an intentionally-sparsified graph can contain a lot of missing links from the testing set, leading to corrupted local neighborhoods of links which cannot reflect the real environments surrounding. FakeEdge does not have the potential to alleviate such a dataset shift. We leave this as a future work.

TABLE A.4

MODEL PERFORMANCE WITH ONLY 20% TRAINING DATA (AUC).

Models	Fake Edge	Cora	Citeseer	Pubmed	USAir	NS	PB	Yeast	C.ele	Power	Router	E.coli
GCN	*Original*	55.35±1.07	56.02±0.94	57.09±1.46	80.44±1.44	61.14±1.29	88.54±0.34	81.09±0.53	66.67±2.33	49.18±0.53	62.98±8.98	81.79±0.76
	Edge Plus	**64.19±0.60**	**62.20±1.07**	**85.65±0.26**	**88.34±0.66**	**62.97±1.56**	92.40±0.26	**85.56±0.26**	**71.21±1.26**	**52.77±1.78**	77.86±0.98	91.68±0.27
	Edge Minus	62.89±0.82	61.47±0.87	85.48±0.28	86.46±1.53	62.17±1.28	**92.63±0.26**	85.44±0.64	70.15±1.14	52.66±1.26	77.68±0.81	**91.86±0.22**
	Edge Mean	63.36±0.75	61.76±1.00	85.64±0.29	87.47±1.18	62.91±1.29	92.52±0.25	84.99±0.47	70.84±1.36	51.96±1.30	**77.96±0.88**	91.81±0.20
	Edge Att	63.30±1.17	61.89±0.90	85.55±0.29	88.19±1.26	61.84±1.77	92.57±0.24	85.51±0.40	70.28±1.42	52.38±1.53	77.44±0.64	91.60±0.33
SAGE	*Original*	51.47±1.68	54.02±1.63	57.00±7.69	84.38±1.14	62.54±1.48	89.48±0.46	77.99±1.05	66.06±2.01	51.73±0.80	61.14±8.58	85.01±0.67
	Edge Plus	65.01±0.61	**63.10±0.63**	**86.90±0.25**	**89.17±0.80**	**65.19±1.75**	**92.71±0.27**	**86.74±0.38**	**72.10±1.81**	**53.99±0.70**	78.72±0.65	91.92±0.15
	Edge Minus	62.71±0.79	58.70±1.00	85.51±0.21	87.24±0.83	63.64±1.52	91.95±0.30	85.61±0.49	70.45±1.98	52.28±0.68	70.66±0.95	89.56±0.32
	Edge Mean	64.44±0.82	62.63±0.63	86.54±0.13	88.95±0.72	64.33±1.48	92.67±0.26	86.60±0.30	71.78±1.80	52.38±0.70	78.54±0.70	91.86±0.19
	Edge Att	**65.31±0.63**	62.81±0.94	86.62±0.24	88.73±0.62	63.90±1.77	92.70±0.22	86.64±0.34	71.40±1.54	53.07±1.32	**79.08±0.61**	**92.14±0.20**
GIN	*Original*	61.93±0.93	61.27±0.95	74.32±1.30	87.39±0.71	62.70±0.81	89.52±0.29	81.70±0.95	68.25±1.65	52.02±1.20	76.50±0.95	90.07±0.51
	Edge Plus	**63.30±0.84**	**62.64±1.17**	**86.53±1.08**	87.63±0.78	65.28±1.44	91.69±0.35	86.30±0.51	69.82±1.47	53.55±0.92	78.08±0.98	91.44±0.35
	Edge Minus	62.66±1.08	62.11±1.17	85.12±0.29	87.31±0.85	65.01±1.91	**91.75±0.33**	86.25±0.58	69.72±1.27	53.59±0.92	78.09±0.93	91.39±0.22
	Edge Mean	62.82±1.33	62.40±1.10	85.67±0.66	87.46±0.75	65.02±1.65	91.72±0.31	86.29±0.53	69.94±1.53	53.54±0.84	78.09±1.08	91.36±0.26
	Edge Att	63.25±0.70	62.07±0.65	85.37±0.64	**87.75±0.92**	**65.54±1.23**	91.62±0.11	**86.31±0.44**	**71.47±1.39**	**54.05±0.85**	**78.79±0.66**	**91.49±0.17**
PLNLP	*Original*	63.08±1.01	65.23±1.41	73.97±1.58	84.14±1.94	63.06±1.30	88.34±1.22	81.15±1.52	68.33±1.70	52.27±0.83	68.90±1.08	84.80±1.72
	Edge Plus	70.81±0.70	72.30±1.59	**94.57±0.15**	89.47±0.87	**65.61±0.80**	**92.66±0.22**	86.80±0.37	**73.35±1.44**	**54.19±0.74**	79.07±0.60	**92.00±0.29**
	Edge Minus	67.21±1.56	68.79±1.12	93.77±0.13	87.49±1.29	64.13±1.79	92.22±0.24	85.78±0.46	71.22±1.60	52.92±0.70	75.97±0.60	91.08±0.23
	Edge Mean	**72.01±0.96**	**72.79±1.72**	94.23±0.22	**89.58±0.89**	65.33±0.68	92.63±0.22	86.80±0.43	73.07±1.39	54.13±0.50	**80.38±1.10**	91.93±0.29
	Edge Att	71.62±1.28	72.72±0.90	94.27±0.22	89.49±0.81	65.58±0.84	92.63±0.21	**86.80±0.40**	73.08±1.41	54.06±0.54	80.37±0.71	91.98±0.26
SEAL	*Original*	58.94±1.72	59.12±0.76	78.00±1.94	87.27±1.30	62.59±1.06	91.32±0.30	83.13±0.83	69.25±1.30	51.02±0.83	71.88±1.43	90.89±0.41
	Edge Plus	**62.62±1.16**	**62.38±0.91**	85.31±0.36	87.62±0.84	63.31±1.41	91.67±0.18	**85.74±0.62**	69.29±0.97	52.86±0.62	77.79±0.66	91.36±0.43
	Edge Minus	60.75±1.88	61.58±0.76	**86.03±1.31**	87.54±1.23	63.84±0.89	91.64±0.21	85.43±0.63	69.16±1.01	52.77±0.86	**78.06±0.80**	91.31±0.45
	Edge Mean	61.55±2.03	62.19±0.56	85.50±0.48	87.52±1.00	63.27±1.18	91.68±0.21	85.64±0.56	**69.30±1.22**	52.40±0.61	77.57±0.90	**91.41±0.43**
	Edge Att	61.77±1.89	62.15±0.53	85.09±0.53	**87.76±0.73**	**64.01±1.69**	**91.91±0.36**	85.56±0.54	68.97±1.24	**53.11±0.95**	77.82±0.54	91.01±0.32
WalkPool	*Original*	64.05±0.74	63.51±1.35	86.85±0.83	94.54±0.87	90.70±0.78	93.45±0.36	94.26±0.63	87.18±0.79	65.89±0.76	83.63±4.29	91.95±1.48
	Edge Plus	64.15±0.87	63.49±1.08	91.73±0.38	94.55±0.92	90.57±0.86	94.39±0.14	94.81±0.24	**87.22±0.77**	66.74±0.54	87.53±0.40	95.30±0.16
	Edge Minus	**64.21±0.97**	63.53±1.04	**91.81±0.87**	94.65±0.89	90.69±0.86	94.39±0.14	94.83±0.22	87.20±0.76	66.77±0.65	87.60±0.38	95.30±0.15
	Edge Mean	64.13±0.89	63.47±1.59	91.46±0.86	94.65±0.96	**90.71±0.86**	**94.40±0.13**	94.83±0.21	87.13±0.77	**66.79±0.61**	87.57±0.39	95.32±0.16
	Edge Att	63.90±1.21	**63.62±1.39**	90.97±1.62	**94.71±0.80**	90.69±0.81	94.40±0.14	**94.85±0.24**	87.17±0.82	66.78±0.54	**87.66±0.29**	**95.32±0.13**

A.7 Concatenation as another valid Edge Invariant subgraph embedding

Edge Concat To fuse the feature from *Edge Plus* and *Edge Minus*, another simple and intuitive way is to concatenate two embedding into one representation. Namely, $\mathbf{h}^{concat} = [\mathbf{h}^{plus}; \mathbf{h}^{minus}]$, where $[\cdot;\cdot]$ is the concatenation operation. $\mathbf{h}^{concat}$ is also an Edge Invariant subgraph embedding. In Table A.5, we observe that *Edge Concat* has the similar performance improvement like other FakeEdge methods on all different backbone models.

TABLE A.5

COMPARISON FOR CONCATENATION OPERATION (AUC)

Models	Fake Edge	Cora	Citeseer	Pubmed	USAir	NS	PB	Yeast	C.ele	Power	Router	E.coli
GCN	*Original*	84.92±1.95	77.05±2.18	81.58±4.62	94.07±1.50	96.92±0.73	93.17±0.45	93.76±0.65	88.78±1.85	76.32±4.65	60.72±5.88	95.35±0.36
	Edge Att	92.06±0.85	88.96±1.05	97.96±0.12	97.20±0.69	97.96±0.39	**95.46±0.45**	**97.65±0.17**	89.76±2.06	85.26±1.32	95.90±0.47	98.04±0.16
	Edge Concat	**92.63±1.00**	**89.88±1.00**	**97.96±0.11**	**97.27±0.95**	**98.07±0.78**	95.39±0.44	97.55±0.46	**89.78±1.59**	**85.71±0.75**	**96.19±0.59**	**98.06±0.23**
SAGE	*Original*	89.12±0.90	87.76±0.97	94.95±0.44	96.57±0.57	98.11±0.48	94.12±0.45	97.11±0.31	87.62±1.63	79.35±1.66	88.37±1.46	95.70±0.44
	Edge Att	**93.31±1.02**	91.01±1.14	98.01±0.13	97.40±0.94	**98.70±0.59**	95.49±0.49	**98.22±0.24**	90.64±1.88	**86.46±0.91**	**96.31±0.59**	**98.43±0.13**
	Edge Concat	93.03±0.57	**91.14±1.46**	**98.08±0.07**	**97.54±0.70**	98.59±0.26	**95.66±0.39**	98.04±0.37	**91.14±1.19**	86.46±1.04	96.19±0.54	98.40±0.22
GIN	*Original*	82.70±1.93	77.85±2.64	91.32±1.13	94.89±0.89	96.05±1.10	92.95±0.51	94.50±0.65	85.23±2.56	73.29±3.88	84.29±1.20	94.34±0.57
	Edge Att	90.76±0.88	89.55±0.61	**97.50±0.15**	**96.34±0.82**	98.35±0.54	95.29±0.29	97.66±0.33	**89.39±1.61**	86.21±0.67	**95.78±0.52**	**97.74±0.33**
	Edge Concat	**90.90±0.92**	**89.94±0.89**	97.48±0.16	96.17±0.64	**98.41±0.73**	**95.45±0.39**	**97.71±0.38**	88.81±1.41	**86.77±0.99**	95.72±0.47	97.72±0.18
PLNLP	*Original*	82.37±1.70	82.93±1.73	87.36±4.90	95.37±0.87	97.86±0.93	92.99±0.71	95.09±1.47	88.31±2.21	81.59±4.31	86.41±1.63	90.63±1.68
	Edge Att	91.22±1.34	88.75±1.70	98.41±0.17	**98.13±0.61**	98.70±0.40	**95.32±0.38**	**98.06±0.37**	91.72±2.12	**90.08±0.54**	**96.40±0.40**	98.01±0.18
	Edge Concat	**93.01±1.16**	**91.19±1.52**	**98.45±0.12**	97.86±0.37	**98.81±0.33**	95.18±0.24	98.04±0.21	**91.79±1.79**	89.16±1.01	96.31±0.36	**98.13±0.18**
SEAL	*Original*	90.13±1.94	87.59±1.57	95.79±0.78	**97.26±0.58**	97.44±1.07	95.06±0.46	96.91±0.45	88.75±1.90	78.14±3.14	92.35±1.21	97.33±0.28
	Edge Att	**91.08±1.67**	89.35±1.43	97.26±0.45	97.04±0.79	**98.52±0.57**	95.19±0.43	**97.70±0.40**	**89.37±1.40**	85.24±1.39	95.14±0.62	**97.90±0.33**
	Edge Concat	90.22±1.60	**89.93±1.31**	**97.40±0.24**	96.83±1.01	98.23±0.49	**95.29±0.43**	97.68±0.34	88.99±1.13	**85.60±1.03**	**95.76±0.74**	97.72±0.25
WalkPool	*Original*	**92.00±0.79**	89.64±1.01	97.70±0.19	97.83±0.97	99.00±0.45	94.53±0.44	96.81±0.92	93.71±1.11	82.43±3.57	87.46±7.45	95.00±0.90
	Edge Att	91.98±0.80	89.36±0.74	98.37±0.19	**98.12±0.81**	99.03±0.50	**95.47±0.27**	98.28±0.24	93.63±1.11	91.25±0.60	97.27±0.27	**98.70±0.14**
	Edge Concat	91.77±1.06	**89.79±0.87**	**98.48±0.09**	98.07±0.86	**99.05±0.44**	95.46±0.35	**98.30±0.25**	**93.82±1.09**	**91.29±0.77**	**97.31±0.27**	98.70±0.17

A.8 Dataset shift vs expressiveness: which contributes more with FakeEdge?

In Section 2.5.3, we discussed how FakeEdge can enhance the expressive power of GNN-based models on non-isomorphic focal node pairs. Meanwhile, we have witnessed the boost of model performance brought by FakeEdge in the experiments. One natural question to ask is whether resolving the dataset shift issue or lifting up expressiveness is the major contributor to make the model perform better.

To answer the question, we first revisit the condition of achieving greater expressiveness. FakeEdge will lift up the expressive power when there exists two nodes being isomorphic in the graph, where we can construct a pair of non-isomorphic focal node pairs which GNNs cannot distinguish. Therefore, how often such isomorphic nodes exist in a graph will determine how much improvement FakeEdge can make by bringing greater expressiveness. Even though isomorphic nodes are common in specific types of graphs like regular graphs, it can be rare in the real world datasets [154]. Thus, we tend to conclude that the effect of solving dataset shift issue by FakeEdge contributes more to the performance improvement rather than greater expressive power. But fully answering the question needs a further rigorous study.

A.9 Limitation

FakeEdge can align the embedding of isomorphic subgraphs in training and testing sets. However, it can pose a limitation that hinders one aspect of the GNN-based model's expressive power. Figure 2.1 gives an example where subgraphs are from training and testing phases, respectively. Now consider that those two subgraphs are both from training set (c = train). Still, the top subgraph has edge observed at focal node pair (y = 1), while the other does not (y = 0). With FakeEdge, two subgraphs will be modified to be isomorphic, yielding the same representation. However, they are non-isomorphic before the modification. To the best of our knowledge, no existing

method can simultaneously achieve the most expressive power and get rid of dataset shift issue, because the edge at the focal node pair in the testing set can never be observed under a practical problem setting.

APPENDIX B

CORE: DATA AUGMENTATION FOR LINK PREDICTION VIA INFORMATION BOTTLENECK

B.1 Related works

Data augmentation. The efficacy of data augmentation strategies in enhancing model generalization is well-documented across various domains. Traditional DA techniques, such as oversampling, undersampling, and interpolation methods [24], have proven to be instrumental in mitigating issues related to learning from imbalanced datasets. In recent years, DA has found extensive application in computer vision and natural language processing. Within the realm of computer vision, techniques such as horizontal flipping, random erasing [189], Hide-and-Seek [135], and Cutout [35] have demonstrated their ability to bolster model performance. On the other hand, in natural language processing, DA is often achieved through lexical substitution strategies, where words are replaced with their semantically equivalent counterparts [184]. UDA [166] introduces a novel approach that leverages TF-IDF metrics for keyword feature augmentation in documents.

Graph data augmentation. Graph-structured data, with its heterogeneous information modalities and complex properties, presents a more intricate landscape for DA compared to conventional image or text data. Typically, graph data augmentation can be bifurcated into two primary approaches: perturbation of graph structure and enhancement of node attributes.

In the realm of semi-supervised node classification, several innovative techniques have been proposed. Drop Edge [127], for instance, introduces random edge dropping to mitigate the oversmoothing problem prevalent in GNNs. Similarly, SDRF [145] leverages graph structure rewiring to address the over-squashing issue in GNNs. NodeAug [156] proposes a more holistic approach by simultaneously augmenting both the graph structure (via edge addition/deletion) and node attributes (via feature replacement).

As for the graph classification task, AD-GCL [140] pioneers an adversarial augmentation technique to boost the training of graph contrastive learning. Concurrently, JOAO [174] and GraphAug [94] automates the selection of augmentations from a predefined pool, incorporating both edge perturbation and node attribute masking.

However, the domain of link prediction has seen relatively limited exploration of DA. Notable exceptions include Distance Encoding [90] and Node labeling [182], which enhance GNNs' expressiveness by infusing distance information. Moreover, Hwang et al. [67] proposes to improve both model expressiveness and node impact by incorporating a virtual node as a message-passing hub for link prediction.

Information bottleneck principle. The Information Bottleneck (IB) principle has been increasingly incorporated into deep learning models to enhance learning robustness. DeepVIB [6], for instance, fistly introduces the application of the IB principle in this domain. To overcome the intractable computation posed by the mutual information term in IB, DeepVIB devises a variational approximation akin to the approach used in Variational Autoencoders (VAEs) [79]. The principle of IB has also been leveraged within the realm of graph representation learning. GIB [163] was among the first to integrate IB into graph learning, aiming to protect GNNs from adversarial attacks. VIB-GSL [138] expanded upon this by applying the IB princi-

ple to graph structure learning, demonstrating its robustness in graph classification tasks. GSAT [103], which is the most related to our work, employs IB to extract the most rationale components from graphs for interpretation purposes. Similarly, IB-subgraph [175] uses IB in conjunction with a bi-level optimization process to identify the most representative subgraph components.

B.1.1 Comparison to GSAT

Scope & Objective. While GSAT [103] has significantly influenced our research, our study introduces a novel graph DA method designed specifically for link prediction tasks, distinguishing itself from GSAT's focus on graph classification interpretability.

Strategic Design. Unlike the direct application of GIB/GSAT to link prediction, we strategically designed CORE's components:

- **Isolated DA Design**: GSAT when directly applied to the link prediction task resulted in conflicts between optimal DAs for varying target links. We address this by adopting a subgraph-based approach for isolating DA effects per target link. Figure 3.4 in Section 3.5.3 showcases how the learned edge masking differs for various downstream target links. The frequent occurrence of larger standard deviations of edge maskings of the same edge for different target links implies substantial disagreement on the optimal DAs. This verifies the necessity of our subgraph-based approach for isolating DA effects per target link.
- **Complete before Reduce**: Recognizing that traditional IB-based methods often overlook data instance information recovery prior to compression, we introduced Complete stage. By recovering missing data, the complete stage can attempt to fulfill the critical assumption (2) in Theorem 3 so that the final DA is predictive and concise. The performance comparison in Table 3.1 necessitates the need for the Complete stage.

Furthermore, we introduced several improvements to achieve better performance and stability (See the ablation study in Appendix B.3.3):

- **Attention Mechanism**: When pruning the noisy edges, GIB/GSAT assumes that knowing the edge representation itself is sufficient to determine whether it is a critical substructure. However, in our link prediction task, one edge may

be critical for one target link but not for another. We propose using target link representation to discern the criticality of an edge for the said link with attention mechanism, which contrasts with GSAT's approach. (Equation 3.10)

- **Edge Label**: To differentiate original edges from those introduced in the complete stage, we have adopted a labeling strategy based on their scores to discern their relative importance. (Appendix B.2.2)
- **Unbiased KL Loss**: In GSAT implementation, the KL loss term is not an unbiased empirical estimator. In a mini-batch B during training, the KL loss term in GSAT is minimized through $\mathbb{E}_G[\mathrm{KL}] \approx \frac{\sum_{G\in B}\mathrm{KL}}{\sum_{G\in B}|E_G|}$. That is, their minimization treats each edge in the batch of graphs as a data instance. However, the unbiasd estimator should be $\frac{\sum_{G\in B}\mathrm{KL}}{|B|}$,, which average across the subgraphs. The KL loss estimator in GSAT is not that troublesome because the number of edges in their benchmark graphs is similar. However, the number of edges in the subgraphs of the target link varies a lot. The performance comparison in the ablation study necessitates the unbiased KL loss estimator.

Distinct Utility.

- **Transferability of Graph Structures**: Our method goes beyond previous IB-based graph ML approaches by verifying that our refined graph structures can serve as inputs to other link prediction models like CN, AA or RA. (See Section 4.2)
- **Robustness to Adversarial Attack**: Our method exhibits robustness to adversarial perturbations targeted at link prediction tasks, further validating the capability of our method to capture critical substructures. (See Section 4.2)

In essence, while GSAT inspired our work, our contributions are substantial, addressing nuances of the link prediction task and refining the DA approach for better performance. The ablation studies (Appendix B.3.3) suggest that applying IB upon link prediction task is a non-trivial work. It requires a dedicated design to suit the unique challenge of link prediction. These distinctions highlight CORE's value and uniqueness in the graph learning domain.

B.2 Implementation details

B.2.1 Marginal distribution

We discuss the marginal distribution term $r(G^{\pm}_{(i,j)})$ in Equation 3.5. Since $\tilde{G}_{(i,j)}$ is sampled based on $G^{+}_{(i,j)}$ through $\tilde{\omega}_{(u,v)} \sim \text{Bern}(\gamma)$, we can write $r(G^{\pm}_{(i,j)}) = \sum_{G^{+}_{(i,j)}} \mathbb{P}(\tilde{\omega}|G^{+}_{(i,j)})\mathbb{P}(G^{+}_{(i,j)})$. Because $\tilde{\omega}$ is independent from the the inflated graph $G^{+}_{(i,j)}$ given its size $n^{\pm}$, $r(G^{\pm}_{(i,j)}) = \sum_{n^{\pm}} \mathbb{P}(\tilde{\omega}|n^{\pm})\mathbb{P}(n^{\pm}) = \mathbb{P}(n^{\pm}) \prod_{u,v} \mathbb{P}(\tilde{\omega}_{(u,v)})$. Thus, the KL-divergence term in Equation 3.6 can be written as:

$$\begin{aligned}
&\text{KL}(p_\phi(G^{\pm}_{(i,j)}|G^{+}_{(i,j)})||r(G^{\pm}_{(i,j)})) = \\
&\sum_{(u,v)\in E^{+}_{(i,j)}} p_{(u,v)} \log \frac{p_{(u,v)}}{\gamma} + \big(1 - p_{(u,v)}\big) \log \frac{1 - p_{(u,v)}}{1 - \gamma} \\
&+ \text{Constant},
\end{aligned} \tag{B.1}$$

where $p_{(u,v)} = \text{sigmoid}(a_{(u,v)})$ and the constant term accounts for the terms of node pairs $(u, v) \notin E^{+}_{(i,j)}$ without any trainable parameters.

In practice, we further allow the constraint hyperparameter γ to be different for the original edges in the inflated graph $G^{+}_{(i,j)}$ and those added in the Complete stage. Namely, we have γ_{ori} and γ_{ext} such that:

$$\begin{aligned}
&\text{KL}(p_\phi(G^{\pm}_{(i,j)}|G^{+}_{(i,j)})||r(G^{\pm}_{(i,j)})) = \\
&\sum_{(u,v)\in E^{+}_{(i,j)}\cap E} p_{(u,v)} \log \frac{p_{(u,v)}}{\gamma_{\text{ori}}} + \big(1 - p_{(u,v)}\big) \log \frac{1 - p_{(u,v)}}{1 - \gamma_{\text{ori}}} \\
&+ \sum_{(u,v)\in E^{+}_{(i,j)}\cap E_{ext}} p_{(u,v)} \log \frac{p_{(u,v)}}{\gamma_{\text{ext}}} + \big(1 - p_{(u,v)}\big) \log \frac{1 - p_{(u,v)}}{1 - \gamma_{\text{ext}}} \\
&+ \text{Constant}.
\end{aligned} \tag{B.2}$$

B.2.2 Awareness of edges scores from the Complete stage

During the Reduce stage, when obtaining the edge representation, $\mathbf{h}(u, v)$, we enrich this representation by appending an additional encoding, $\tilde{\mathbf{h}}(u, v)$. This supplementary encoding serves to inform the model about the edge's origin — whether it's a component of the original graph or an edge added in the Complete stage. However, this encoding strategy does not provide insights into the relative importance of the newly added edges.

To address this, we incorporate a ranking mechanism, assigning a rank label to each added edge based on its relative importance. Specifically, we segregate the added edges into ten equal-sized buckets, determined by their respective scores. Each edge is then assigned a label corresponding to its bucket number. This method of score-aware edge representation enables the model to make more informed use of the added edges, and to discern the most informative edges from the others.

B.2.3 Augmentation during inference

During the training phase, CORE reduces the inflated graph by implementing edge sampling. However, for the testing phase, we do not employ sampling to obtain the augmented graph.

Given that the entire model can be conceptualized as a probabilistic model, the inference stage requires us to compute the expectation of the random variables. As such, for each edge (u, v), we use its expected value, $p_{(u,v)}$, as the edge weight during the inference process.

B.2.4 Nodewise sampling

The implementation of CORE applies an attention mechanism on each edge (u, v) to get $a_{(u,v)}$, which can consume a huge amount of GPU memory when operating on large-scale graphs. As a compromised modification, we follow [103] to sample the node

in the inflated graph instead of the edges. More specifically, after we get the node representation, we apply the attention to the node and the subgraph representation to get the importance score a_u for each node. Then, each node is still sampled through a Bernoulli distribution $\omega_u \sim \text{Bern}(\text{sigmoid}(a_u))$. In the end, the edge mask is obtained by $\omega_{(u,v)} = \omega_u * \omega_v$.

B.2.5 Hyperparameter details

In the Complete stage of our experiments, we employ *GCN* and *SAGE* as link predictors to inject potential missing edges into the four non-attributed graphs. This choice is driven by the smaller sizes of these graphs. The parameter k, representing the number of top-scored edges, is searched within the range $[1000, 2000]$.

For the remaining four attributed graphs, we limit our search to heuristic link predictors, namely *CN*, *AA*, and *RA*. The k parameter for these graphs is explored within the range $[16000, 32000]$.

In the Reduce stage, we conduct a hyperparameter search for both γ and β. Specifically, we search for the parameters γ_{ori} and γ_{ext} within the set $[0.8, 0.5, 0.2]$. The parameter β is searched within the range $[1, 0.1, 0.01]$.

B.2.6 Software and hardware details

Our implementation leverages the PyTorch Geometric library [50] and the SEAL [178] framework. All experiments were conducted on a Linux system equipped with an NVIDIA P100 GPU with 16GB of memory.

B.2.7 Time complexity

The time complexity of our method is primarily similar to that of SEAL. However, there are two additional computational requirements: (1) an extra node representation encoding is needed for edge pruning; and (2) a probability score must

be assigned to each edge. Consequently, the overall computation complexity of our method is $O(t(d^{l+1}F'' + d^{l+1}F''^2))$, where t represents the number of target links, d is the maximum node degree, l corresponds to the number of hops of the subgraph, and F'' indicates the dimension of the representation.

B.3 Supplementary experiments

B.3.1 Baseline methods

***CN* [91].** Common Neighbor (CN) is a widely-used link predictor that posits a node pair with more common neighbors is more likely to connect. The score is computed as $CN(i,j) = |\mathcal{N}_i \cap \mathcal{N}_j|$.

***AA* [4].** Adamic-Adar (AA) extends the CN approach, emphasizing that common neighbors with fewer connections are more important than those with many connections. The score is calculated as $AA(i,j) = \sum_{z \in \mathcal{N}_i \cap \mathcal{N}_j} \frac{1}{\log |\mathcal{N}_z|}$.

RA [191]. Resource Allocation (RA) modifies the weight decay of AA on common neighbors. It computes the score as $RA(i,j) = \sum_{z \in \mathcal{N}_i \cap \mathcal{N}_j} \frac{1}{|\mathcal{N}_z|}$.

***GCN* [82].** Graph Convolutional Networks (GCN) propose a graph convolution operation using the first-degree neighbors. Owing to its computational efficiency and high performance, GCN is a popular architecture for GNNs.

***SAGE* [59].** GraphSAGE (SAGE) proposes a scalable approach to applying GNNs on large graphs. The encoder part of SAGE uses two distinct weights for the center node representation and its surrounding neighbors.

***GIN*[167].** Graph Isomorphism Network (GIN) is a 1-WL expressive GNN widely used for graph classification problems. In our experiment, we use the zero-one labeling

trick[182] in GIN to distinguish between the target node pair and the remaining nodes in the target link's local neighborhood.

***SEAL*[178].** SEAL is a state-of-the-art link prediction model. It uses GNNs and a node labeling trick[182] to enhance expressiveness for link prediction. We found that SEAL's expected performance on the **Collab** dataset is approximately 5% higher than initially reported.

***CFLP* [185].** CFLP introduces counterfactual links into the graph to enable causal inference for link prediction. Despite its efficiency for smaller graphs, CFLP can cause out-of-memory issues for larger graphs due to its preprocessing step to find counterfactual node pairs.

***Edge Proposal* [134].** Edge Proposal augments the graph by adding potential missing or future links to complement the graph for enhanced link prediction performance.

***Node Drop* [120].** Node Drop, also known as DropGNN, randomly drops nodes in the graph. This exposes the model to multiple views of the graph, enhancing its expressiveness.

***Edge Drop* [127].** Edge Drop, also known as DropEdge in their work, introduces a stochastic approach to edge removal as a regularization method to solve the over-smoothing issue in GNNs.

B.3.2 Benchmark datasets

The following graph datasets were utilized in our experiments:

Non-attributed graph datasets:

1. **USAir**[10]: This dataset contains a representation of US airlines, encapsulating the connectivity between different airports.
2. **Yeast**[150]: This dataset includes a protein-protein interaction network within yeast cells, providing insights into the complex interplay of biological components.
3. **C.ele**[158]: This dataset represents the neural network of the nematode Caenorhabditis elegans, one of the most studied organisms in neuroscience.
4. **Router**[136]: This dataset encompasses Internet connectivity at the router-level, providing a snapshot of the web's underlying infrastructure.

Attributed graph datasets:

1. **CS**[131]: This dataset provides a snapshot of the collaboration network in the computer science domain, highlighting co-authorship relationships.
2. **Physics**[131]: This dataset depicts a collaboration network within the field of physics, offering insights into academic partnerships.
3. **Computers**[131]: This dataset presents a segment of the co-purchase network on Amazon, reflecting purchasing behavior related to computer products.
4. **Collab**[163]: This dataset presents a large-scale collaboration network, showcasing a wide array of interdisciplinary partnerships.

Comprehensive statistics for these datasets are detailed in Table B.1. Note that when we perform the train test split, we ensure that the split is the same for all different methods on each dataset.

B.3.3 More ablation study

We further conduct ablation studies on three unique components we specifically design for link prediction tasks, to enhance the performance of our proposed methods. The results are presented in Table B.3. As the results suggest, all three components can boost the performance of the proposed DA method by varying degrees. For instance, CORE with *No Attention* hampers the performance from 0.6 to 2.5 in Hits@50. CORE with *No Edge Label* also drops the effectiveness of DAs by from 0.5

TABLE B.1

STATISTICS OF BENCHMARK DATASETS.

Dataset	#Nodes	#Edges	Avg. node deg.	Max. node deg.	Density	Attr. Dimension
C.ele	297	4296	14.46	134	9.7734%	-
USAir	332	4252	12.81	139	7.7385%	-
Yeast	2375	23386	9.85	118	0.8295%	-
Router	5022	12516	2.49	106	0.0993%	-
CS	18333	163788	8.93	136	0.0975%	6805
Physics	34493	495924	14.38	382	0.0834%	8415
Computers	13752	491722	35.76	2992	0.5201%	767
Collab	235868	2358104	10.00	671	0.0085%	128

to 3. The *Biased KL Loss* mostly hurts the performance of CORE on **USAir** and **Physics**, which we assume that the degree distribution of these two datasets is more skewed compared to others.

B.3.4 Parameter sensitivity

We also conducted an experiment to examine the sensitivity of the hyperparameters in CORE. We focused on the Reduce stage only, as this is the core component of our method. As per our hyperparameter search procedure, we evaluated the model performance for β across $[10, 5, 2, 1, 0.1, 0.01]$ and γ across $[0.8, 0.5, 0.2]$. The results are depicted in Figure B.1. Our method consistently enhanced the model performance across various hyperparameter settings. However, we also notice that the performance of CORE will drop if we increase the graph compression term β or decrease the edge-preserving term γ. This is because sparsifying the graph too much will result in the loss of critical information. Thus, balanced β and γ are crucial for a robust CORE performance.

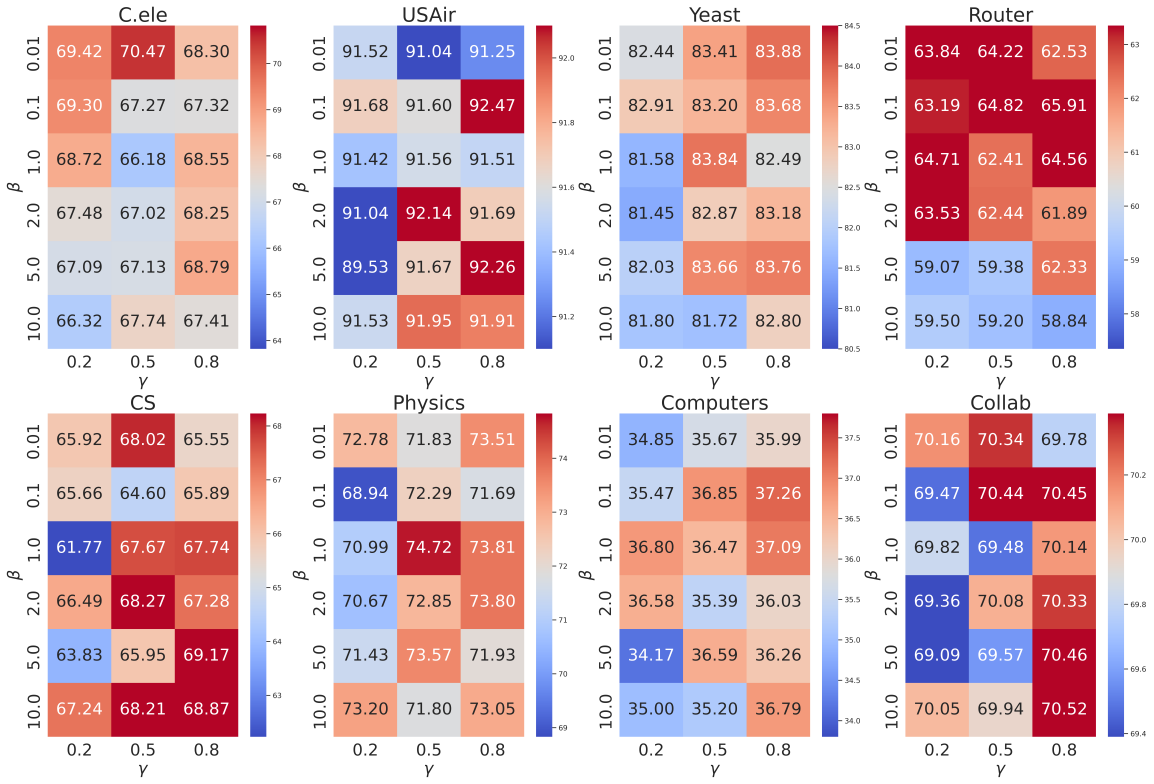

Figure B.1: CORE can improve LP performance in various hyperparameter settings measured by Hits@50. Warmer colors indicate improved performance over the baseline, whereas cooler colors signify the contrary.

TABLE B.2

RESULTS OF ADVERSARIAL ROBUSTNESS.

Methods	Yeast				Router				CS				Physics				Computers			
	No Adv	10%	30%	50%	No Adv	10%	30%	50%	No Adv	10%	30%	50%	No Adv	10%	30%	50%	No Adv	10%	30%	50%
GCN	80.33	76.69	68.30	57.96	35.16	28.55	20.75	15.86	60.69	64.24	57.49	45.28	69.16	68.85	60.16	46.76	32.70	30.16	24.33	20.07
SAGE	78.34	74.33	67.01	56.23	35.76	35.13	29.01	25.11	31.44	53.92	42.72	29.16	22.87	61.69	50.20	36.83	14.53	10.42	4.16	4.34
ELPH	78.92	76.74	69.05	65.58	59.50	57.07	52.83	47.65	67.84	65.92	60.28	50.09	69.60	64.67	58.83	47.51	33.64	34.30	29.35	23.23
NCNC	73.11	71.34	66.34	59.60	57.13	55.04	52.01	47.47	65.73	63.13	53.93	36.66	72.87	69.27	58.74	45.31	37.17	35.74	33.59	31.53
SEAL	82.50	78.24	69.24	62.84	60.35	51.84	48.75	44.02	65.23	60.31	58.97	42.38	71.83	64.28	59.84	44.17	35.80	33.84	31.84	28.84
CORE	**84.67**	**81.94**	**74.70**	**68.96**	**65.64**	**59.20**	**56.58**	**51.02**	**69.67**	**66.87**	**61.18**	**50.49**	**74.73**	**70.78**	**62.58**	**50.37**	**37.88**	**36.28**	**34.66**	**31.85**

Results of adversarial robustness for different models on the rest of datasets. The attack rates of 10%, 30%, and 50% represent the respective ratios of edges subjected to adversarial flips by CLGA [183].

TABLE B.3

ABLATION STUDY EVALUATED BY HITS@50.

Models	**C.ele**	**USAir**	**Router**	**Yeast**	**CS**	**Physics**	**Computers**	**Collab**
No Attention (Equation 3.10)	$74.41_{\pm 1.75}$	$\underline{92.38}_{\pm 1.07}$	$63.60_{\pm 2.92}$	$82.81_{\pm 0.94}$	$65.69_{\pm 2.38}$	$\underline{73.10}_{\pm 0.78}$	$36.85_{\pm 1.17}$	$\underline{70.09}_{\pm 0.94}$
No Edge Label (Appendix B.2.2)	$74.27_{\pm 4.73}$	$91.49_{\pm 0.81}$	$63.07_{\pm 3.90}$	$\underline{83.79}_{\pm 2.01}$	$\underline{65.84}_{\pm 1.40}$	$71.71_{\pm 2.72}$	$36.79_{\pm 2.88}$	$69.64_{\pm 1.32}$
Biased KL Loss	$\underline{75.14}_{\pm 2.18}$	$90.66_{\pm 3.69}$	$\underline{64.80}_{\pm 2.25}$	$81.91_{\pm 1.70}$	$64.22_{\pm 3.38}$	$69.32_{\pm 5.05}$	$\underline{37.31}_{\pm 2.47}$	$69.61_{\pm 1.13}$
CORE	$\mathbf{76.34}_{\pm \mathbf{1.65}}$	$\mathbf{92.69}_{\pm \mathbf{0.75}}$	$\mathbf{65.47}_{\pm \mathbf{2.44}}$	$\mathbf{84.22}_{\pm \mathbf{1.58}}$	$\mathbf{68.15}_{\pm \mathbf{0.78}}$	$\mathbf{74.73}_{\pm \mathbf{2.12}}$	$\mathbf{37.88}_{\pm \mathbf{1.10}}$	$\mathbf{70.64}_{\pm \mathbf{0.51}}$

The **best-performing** components are highlighted in bold, while the second-best performance is underlined.

Besides, the parameter γ acts as a regulatory mechanism that influences each edge's probability, nudging it towards the behavior of a random graph. A lower γ tends to suppress non-essential edges in predictions. Based on our observations, values between $[0.5, 0.8]$ tend to be optimal. That being said, in our studies, the balancing factor β has exhibited a more pronounced effect on model performance than γ.

B.3.5 CORE with GCN and SAGE as backbones

We further investigate CORE with GCN and SAGE as backbones. The results are presented in Table B.4. It shows that CORE can consistently improve model performance with various GNNs as the backbones.

B.3.6 Complete stage only considering node pairs with common neighbors

When we score the potential node pairs to be added into the graph at Complete stage, we only consider those with at least one common neighbor. While this seems to limit the capability of recovering missing edges, it is actually an effective approach with balanced computational efficiency. We empirically investigate the number of node pairs in the testing set that have at least one common neighbor, presented in Table B.5.

With the exception of the Router dataset, our benchmarks consistently indicate that positive testing edges are inclined to have at least one common neighbor. Therefore, our scoring process at the Complete stage encompasses the majority of the testing edges. This observation underscores that our edge addition strategy aligns well with the community-like nature observed in real-world datasets.

B.4 Variational bounds for the GIB objective in Equation 3.4 and Equation 3.5

The objective from Equation 3.3 is:

$$-I(G^{\pm}_{(i,j)}, Y) + \beta I(G^{\pm}_{(i,j)}, G^{+}_{(i,j)}).$$

Since those two terms are computationally intractable, we introduce two variational bounds.

For $I(G^{\pm}_{(i,j)}, Y)$, by the definition of mutual information:

$$\begin{aligned} I(G^{\pm}_{(i,j)}, Y) &= \mathbb{E}[\log \frac{p(Y|G^{\pm}_{(i,j)})}{p(Y)}] \\ &= \mathbb{E}[\log \frac{q_\theta(Y|G^{\pm}_{(i,j)})}{p(Y)}] \\ &+ \mathbb{E}[\mathrm{KL}(p(Y|G^{\pm}_{(i,j)})||q_\theta(Y|G^{\pm}_{(i,j)}))] \\ &\geq \mathbb{E}[\log \frac{q_\theta(Y|G^{\pm}_{(i,j)})}{p(Y)}] \\ &= \mathbb{E}[\log q_\theta(Y|G^{\pm}_{(i,j)})] + H(Y), \end{aligned}$$

where the KL-divergence term is nonnegative.

For the second term $I(G^{\pm}_{(i,j)}, G^{+}_{(i,j)})$, by definition:

$$\begin{aligned} I(G^{\pm}_{(i,j)}, G^{+}_{(i,j)}) &= \mathbb{E}[\log \frac{p(G^{\pm}_{(i,j)}|G^{+}_{(i,j)})}{p(G^{\pm}_{(i,j)})}] \\ &= \mathbb{E}[\log \frac{p_\phi(G^{\pm}_{(i,j)}|G^{+}_{(i,j)})}{r(G^{\pm}_{(i,j)})}] \\ &- \mathbb{E}[\mathrm{KL}(p(G^{\pm}_{(i,j)})||r(G^{\pm}_{(i,j)}))] \\ &\leq \mathbb{E}[\mathrm{KL}(p_\phi(G^{\pm}_{(i,j)}|G^{+}_{(i,j)})||r(G^{\pm}_{(i,j)})]. \end{aligned}$$

B.5 Proof for Theorem 3

We restate Theorem 3: Assume that: (1) The existence Y of a link (i, j) is solely determined by its local neighborhood $G^{*}_{(i,j)}$ in a way such that $p(Y) = f(G^{*}_{(i,j)})$, where f is a deterministic invertible function; (2) The inflated graph contains sufficient structures for prediction $G^{*}_{(i,j)} \in \mathbb{G}_{\mathrm{sub}}(G^{+}_{(i,j)})$. Then $G^{\pm}_{(i,j)} = G^{*}_{(i,j)}$ minimizes the objective in Equation 3.3.

Proof. We can follow a similar derivation as in [103]. Consider the following steps:

$$
\begin{aligned}
& - I(G^{\pm}_{(i,j)}; Y) + \beta I(G^{\pm}_{(i,j)}; G^{+}_{(i,j)}) \\
& \qquad\qquad \text{(Start with the original objective Equation 3.3)} \\
= & - I(G^{+}_{(i,j)}, G^{\pm}_{(i,j)}; Y) + I(G^{+}_{(i,j)}; Y | G^{\pm}_{(i,j)}) + \beta I(G^{\pm}_{(i,j)}; G^{+}_{(i,j)}) \\
& \qquad\qquad \text{(Expand the first term via chain rule of mutual information)} \\
= & - I(G^{+}_{(i,j)}, G^{\pm}_{(i,j)}; Y) + (1-\beta) I(G^{+}_{(i,j)}; Y | G^{\pm}_{(i,j)}) \\
& + \beta I(G^{+}_{(i,j)}; G^{\pm}_{(i,j)}, Y) \\
& \qquad\qquad \text{(Split the third term proportionally)} \\
= & - I(G^{+}_{(i,j)}; Y) + (1-\beta) I(G^{+}_{(i,j)}; Y | G^{\pm}_{(i,j)}) + \beta I(G^{+}_{(i,j)}; G^{\pm}_{(i,j)}, Y) \\
& \qquad\qquad \text{(Because } G^{\pm}_{(i,j)} \text{ is a subgraph of } G^{+}_{(i,j)}\text{)} \\
= & (\beta - 1) I(G^{+}_{(i,j)}; Y) - (\beta - 1) I(G^{+}_{(i,j)}; Y | G^{\pm}_{(i,j)}) \\
& + \beta I(G^{+}_{(i,j)}; G^{\pm}_{(i,j)} | Y), \\
& \qquad\qquad \text{(Split the last term and rearrange terms)}
\end{aligned}
$$

Since $I(G^{+}_{(i,j)}; Y)$ does not involve trainable parameters, we focus on the last two terms. If $\beta \in [0, 1]$, the $G^{\pm}_{(i,j)}$ that minimizes Equation 3.3 also minimizes $-(\beta - 1) I(G^{+}_{(i,j)}; Y | G^{\pm}_{(i,j)}) + \beta I(G^{+}_{(i,j)}; G^{\pm}_{(i,j)} | Y)$. Given that mutual information is nonnegative, the lower bound of $(1-\beta) I(G^{+}_{(i,j)}; Y | G^{\pm}_{(i,j)}) + \beta I(G^{+}_{(i,j)}; G^{\pm}_{(i,j)} | Y)$ is 0.

Next, we show that $G^{\pm}_{(i,j)} = G^{*}_{(i,j)}$ can make Equation 3.3 reach its lower bound. Since $p(Y) = f(G^{*}_{(i,j)})$, $I(G^{+}_{(i,j)}; Y | G^{*}_{(i,j)}) = 0$. That is, there is no mutual information between $G^{+}_{(i,j)}$ and Y when knowing $G^{+}_{(i,j)}; Y$. Similarly, because f is invertible, there is no more mutual information between $G^{+}_{(i,j)}$ and $G^{\pm}_{(i,j)}$ when knowing Y, yielding $I(G^{+}_{(i,j)}; G^{\pm}_{(i,j)} | Y) = 0$. Therefore, $G^{\pm}_{(i,j)} = G^{*}_{(i,j)}$ minimizes Equation 3.3. $\square$

B.6 Limitations

In this section, we address the limitations of our proposed method. Firstly, while our model, CORE, delivers superior performance through the application of distinct augmentations for each link prediction, this practice significantly increases the computational burden. This is due to the requirement of independently calculating the augmentation for each link. Attempts to implement a universal augmentation across all links simultaneously resulted in a significant performance drop. Thus, future work may explore efficient methods to balance computational overhead with performance gains.

Secondly, our backbone model, SEAL, utilizes node labeling to determine proximity to the target link for nodes within the neighborhood. Due to computational constraints, this labeling process is performed on the CPU during preprocessing. Despite the capacity of the Reduce stage to alter the local structure of the target link, the node labels remain unchanged post-graph pruning, potentially leading to information leakage about each node's position in the unaltered graph. Future research could investigate methods to fully conceal this information, enabling link prediction to be purely dependent on the pruned graph.

TABLE B.4

CORE WITH GCN AND SAGE AS BACKBONE MODELS.

Methods	**C.ele**	**USAir**	**Router**	**Yeast**
GCN as the backbone model				
GCN	62.21$_{\pm 6.13}$	83.20$_{\pm 3.88}$	43.37$_{\pm 9.75}$	81.30$_{\pm 1.96}$
Complete Only	65.20$_{\pm 3.76}$	85.84$_{\pm 1.62}$	<u>53.18$_{\pm 6.92}$</u>	81.54$_{\pm 1.10}$
Reduce Only	<u>65.52$_{\pm 2.95}$</u>	<u>86.42$_{\pm 4.03}$</u>	47.26$_{\pm 8.37}$	<u>82.82$_{\pm 0.96}$</u>
CORE	**68.79$_{\pm 2.48}$**	**87.96$_{\pm 1.03}$**	**55.35$_{\pm 4.53}$**	**82.82$_{\pm 0.96}$**
SAGE as the backbone model				
SAGE	70.91$_{\pm 1.05}$	80.38$_{\pm 7.18}$	56.71$_{\pm 2.59}$	84.70$_{\pm 2.01}$
Complete Only	71.59$_{\pm 2.15}$	83.60$_{\pm 2.98}$	58.60$_{\pm 2.79}$	84.91$_{\pm 1.33}$
Reduce Only	<u>72.12$_{\pm 1.84}$</u>	<u>87.32$_{\pm 3.83}$</u>	<u>59.54$_{\pm 2.69}$</u>	<u>85.47$_{\pm 0.96}$</u>
CORE	**73.36$_{\pm 2.08}$**	**89.76$_{\pm 2.25}$**	**61.75$_{\pm 1.07}$**	**85.61$_{\pm 0.98}$**

The **best-performing** methods are highlighted in bold, while the <u>second-best</u> performance is underlined.

TABLE B.5

NUMBER OF NODE PAIRS WITH AT LEAST ONE COMMON NEIGHBOR AS A POSITIVE INSTANCE IN THE TESTING SETS.

Dataset	# Node pairs with CN	# Node pairs	Ratio
C.ele	344	429	80.17%
USAir	387	425	91.05%
Yeast	1700	2338	72.71%
Router	114	1251	9.11%
CS	11306	16378	69.03%
Physics	42061	49592	84.81%
Computers	45676	49172	92.89%

APPENDIX C

PURE MESSAGE PASSING CAN ESTIMATE COMMON NEIGHBOR FOR LINK PREDICTION

C.1 Efficient inference at node-level complexity

In addition to its superior performance, MPLP stands out for its practical advantages in industrial applications due to its node-level inference complexity. This design is akin to employing an MLP as the predictor. Our method facilitates offline preprocessing, allowing for the caching of node signatures or representations. Consequently, during online inference in a production setting, MPLP merely requires fetching the relevant node signatures or representations and processing them through an MLP. This approach significantly streamlines the online inference process, necessitating only node-level space complexity and ensuring constant time complexity for predictions. This efficiency in both space and time makes MPLP particularly suitable for real-world applications where rapid, on-the-fly predictions are crucial.

C.2 Estimate triangular substructures

Not only does MPLP encapsulate the local structure of the target node pair by assessing node counts based on varying shortest-path distances, but it also pioneers in estimating the count of triangles linked to any of the nodes— an ability traditionally deemed unattainable for GNNs [27]. In this section, we discuss a straightforward implementation of the triangle estimation.

C.2.1 Method

Constructing the structural feature with DE can provably enhance the expressiveness of the link prediction model [90, 182]. However, there are still prominent cases where labelling trick also fails to capture. Since labelling trick only considers the relationship between the neighbors and the target node pair, it can sometimes miss the subtleties of intra-neighbor relationships. For example, the nodes of DE$(1,1)$ in Figure 4.4 exhibit different local structures. Nevertheless, labelling trick like DE tends to treat them equally, which makes the model overlook the triangle substructure shown in the neighborhood. [27] discusses the challenge of counting such a substructure with a pure message-passing framework. We next give an implementation of message-passing to approximate triangle counts linked to a target node pair—equivalent in complexity to conventional MPNNs.

For a triangle to form, two nodes must connect with each other and the target node. Key to our methodology is recognizing the obligatory presence of length-1 and length-2 walks to the target node. Thus, according to Theorem 6, our estimation can formalize as:

$$\#(\underset{u}{\triangle}) = \frac{1}{2}\mathbb{E}\Big(\tilde{\boldsymbol{h}}_u^{(1)} \cdot \tilde{\boldsymbol{h}}_u^{(2)}\Big). \tag{C.1}$$

Augmenting the structural features with triangle estimates gives rise to a more expressive structural feature set of MPLP.

C.2.2 Experiments

Following the experiment in Section 6.1 of [27], we conduct an experiment to evaluate MPLP's ability to count triangular substructures. Similarly, we generate two synthetic graphs as the benchmarks: the Erdos-Renyi graphs and the random regular graphs. We also present the performance of baseline models reported in [27].

Please refer to [27] for details about the experimental settings and baseline models. The results are shown in Table C.1.

TABLE C.1

PERFORMANCE OF DIFFERENT GNNS ON LEARNING THE COUNTS OF TRIANGLES.

Dataset	**Erdos-Renyi**	**Random Regular**
GCN	8.27E-1	2.05
GIN	1.25E-1	4.74E-1
SAGE	1.48E-1	5.21E-1
sGNN	1.13E-1	4.43E-1
2-IGN	9.85E-1	5.96E-1
PPGN	2.51E-7	3.71E-5
LRP-1-3	2.49E-4	3.83E-4
Deep LRP-1-3	4.77E-5	5.16E-6
MPLP	1.61E-4	3.70E-4

Performance of different GNNs on learning the counts of triangles, measured by MSE divided by variance of the ground truth counts. Shown here are the median (i.e., third-best) performances of each model over five runs with different random seeds.

As the results show, the triangle estimation component of MPLP can estimate the number of triangles in the graph with almost negligible error, similar to other more expressive models. Moreover, MPLP achieves this with a much lower computational cost, which is comparable to 1-WL GNNs like GCN, GIN, and SAGE. It demonstrates MPLP's advantage of better efficiency over more complex GNNs like 2-IGN and PPGN.

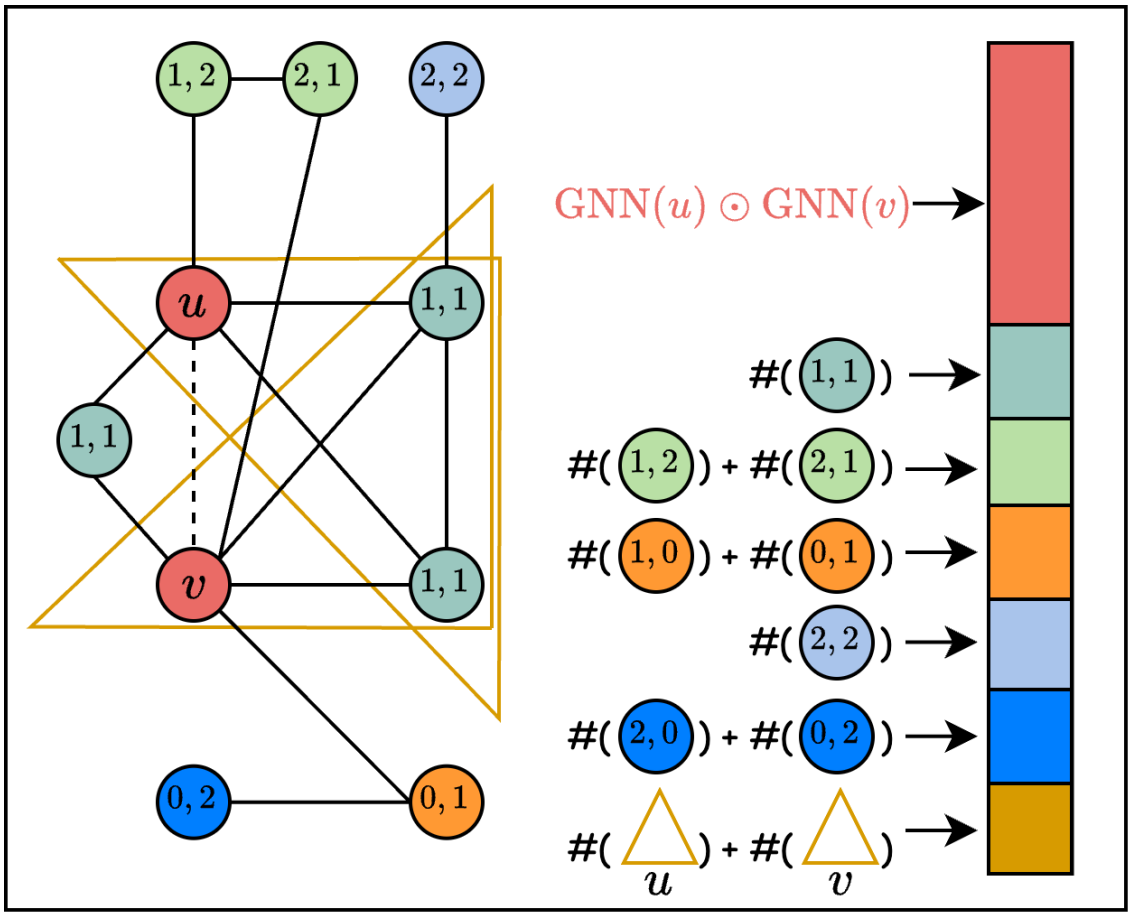

Figure C.1: Representation of the target link (u, v) of MPLP after including the triangular estimation component.

C.3 Experimental details

C.3.1 Benchmark datasets

TABLE C.2

STATISTICS OF BENCHMARK DATASETS.

Dataset	#Nodes	#Edges	Avg. node deg.	Std. node deg.	Max. node deg.	Density	Attr. Dimension
C.ele	297	4296	14.46	12.97	134	9.7734%	-
Yeast	2375	23386	9.85	15.50	118	0.8295%	-
Power	4941	13188	2.67	1.79	19	0.1081%	-
Router	5022	12516	2.49	5.29	106	0.0993%	-
USAir	332	4252	12.81	20.13	139	7.7385%	-
E.coli	1805	29320	16.24	48.38	1030	1.8009%	-
NS	1589	5484	3.45	3.47	34	0.4347%	-
PB	1222	33428	27.36	38.42	351	4.4808%	-
CS	18333	163788	8.93	9.11	136	0.0975%	6805
Physics	34493	495924	14.38	15.57	382	0.0834%	8415
Computers	13752	491722	35.76	70.31	2992	0.5201%	767
Photo	7650	238162	31.13	47.28	1434	0.8140%	745
Collab	235868	2358104	10.00	18.98	671	0.0085%	128
PPA	576289	30326273	52.62	99.73	3241	0.0256%	58
Citation2	2927963	30561187	10.44	42.81	10000	0.0014%	128

The statistics of each benchmark dataset are shown in Table C.2. The benchmarks without attributes are:

- **USAir** [10]: a graph of US airlines;
- **NS** [110]: a collaboration network of network science researchers;

- **PB** [3]: a graph of links between web pages on US political topics;
- **Yeast** [150]: a protein-protein interaction network in yeast;
- **C.ele** [158]: the neural network of Caenorhabditis elegans;
- **Power** [158]: the network of the western US's electric grid;
- **Router** [136]: the Internet connection at the router-level;
- **E.coli** [180]: the reaction network of metabolites in Escherichia coli.

4 out of 7 benchmarks with node attributes come from [131], while Collab, PPA and Citation2 are from Open Graph Benchmark [62]:

- **CS**: co-authorship graphs in the field of computer science, where nodes represent authors, edges represent that two authors collaborated on a paper, and node features indicate the keywords for each author's papers;
- **Physics**: co-authorship graphs in the field of physics with the same node/edge/feature definition as of **CS**;
- **Computers**: a segment of the Amazon co-purchase graph for computer-related equipment, where nodes represent goods, edges represent that two goods are frequently purchased together together, and node features represent the product reviews;
- **Physics**: a segment of the Amazon co-purchase graph for photo-related equipment with the same node/edge/feature definition as of **Computers**;
- **Collab**: a large-scale collaboration network, showcasing a wide array of interdisciplinary partnerships.
- **PPA**: a large-scale protein-protein association network, representing the biological interaction between proteins.
- **Citation2**: a large-scale citation network, with papers as nodes and the citaitons as edges.

Since OGB datasets have a fixed split, no train test split is needed for it. For the other benchmarks, we randomly split the edges into 70-10-20 as train, validation, and test sets. The validation and test sets are not observed in the graph during the entire cycle of training and testing. They are only used for evaluation purposes. For

Collab, it is allowed to use the validation set in the graph when evaluating on the test set.

We run the experiments 10 times on each dataset with different splits. For each run, we cache the split edges and evaluate every model on the same split to ensure a fair comparison. The average score and standard deviation are reported in Hits@100 for PPA, MMR for Citation2 and Hits@50 for the remaining datasets.

C.3.2 More details in baseline methods

In our experiments, we explore advanced variants of the baseline models **ELPH** and **NCNC**. Specifically, for **ELPH**, [22] propose BUDDY, a link prediction method that preprocesses node representations to achieve better efficiency but compromises its expressiveness. **NCNC** [155] builds upon its predecessor, NCN, by first estimating the complete graph structure and then performing inference. In our experiments, we select the most expressiveness variant to make sure it is a fair comparison between different model architectures. Thus, we select **ELPH** over BUDDY, and **NCNC** over NCN to establish robust baselines in our study. We conduct a thorough hyperparameter tuning for **ELPH** and **NCNC** to select the best-performing models on each benchmark dataset. We follow the hyperparameter guideline of **ELPH** and **NCNC** to search for the optimal structures. For **ELPH**, we run through hyperparameters including dropout rates on different model components, learning rate, batch size, and dimension of node embedding. For **NCNC**, we experiment on dropout rates on different model components, learning rates on different model components, batch size, usage of jumping knowledge, type of encoders, and other model-specific terms like alpha. For **Neo-GNN** and **SEAL**, due to their relatively inferior efficiency, we only tune the common hyperparameters like learning rate, size of hidden dimensions.

C.3.3 Evaluation Details: Inference Time

In Figure 4.5, we assess the inference time across different models on the Collab dataset for a single epoch of test links. Specifically, we clock the wall time taken by models to score the complete test set. This encompasses preprocessing, message-passing, and the actual prediction. For the SEAL model, we employ a dynamic subgraph generator during the preprocessing phase, which dynamically computes the subgraph. Meanwhile, for both ELPH and our proposed method, MPLP, we initially propagate the node features and signatures just once at the onset of inference. These are then cached for subsequent scoring sessions.

C.3.4 Software and hardware details

We implement MPLP in Pytorch Geometric framework [50]. We run our experiments on a Linux system equipped with an NVIDIA A100 GPU with 80GB of memory.

C.3.5 Time Complexity

The efficiency of MPLP stands out when it comes to link prediction inference. Let's denote t as the number of target links, d as the maximum node degree, r as the number of hops to compute, and F as the dimension count of node signatures.

For preprocessing node signatures, MPLP involves two primary steps:

1. Initially, the algorithm computes all-pairs unweighted shortest paths across the input graph to acquire the shortest-path neighborhood $\mathcal{N}_v^s$ for each node. This can be achieved using a BFS approach for each node, with a time complexity of $O(|V||E|)$.
2. Following this, MPLP propagates the QO vectors through the shortest-path neighborhood, which has a complexity of $O(td^rF)$, and then caches these vectors in memory.

During online scoring, MPLP performs the inner product operation with a complexity

of $O(tF)$, enabling the extraction of structural feature estimations.

However, during training, the graph's structure might vary depending on the batch of target links due to the shortcut removal operation. As such, MPLP proceeds in three primary steps:

1. Firstly, the algorithm extracts the r-hop induced subgraph corresponding to these t target links. In essence, we deploy a BFS starting at each node of the target links to determine their receptive fields. This process, conceptually similar to message-passing but in a reversed message flow, has a time complexity of $O(tdr)$. Note that, different from SEAL, we extract one r-hop subgraph induced from a batch of target links.
2. To identify the shortest-path neighborhood $\mathcal{N}_v^s$, we simply apply sparse-sparse matrix multiplications of the adjacency matrix to get the s-power adjacency matrix, where $s = 1, 2, \ldots, r$. Due to the sparsity, this takes $O(|V|d^r)$.
3. Finally, the algorithm engages in message-passing to propagate the QO vectors along the shortest-path neighborhoods, with a complexity of $O(td^rF)$, followed by performing the inner product at $O(tF)$.

Summing up, the overall time complexity for MPLP in the training phase stands at $O(tdr + |V|d^r + td^rF)$.

For MPLP+, it does not require the preprocessing step for the shortest-path neighborhood. Thus, the time complexity is the same as any standard message-passing GNNs, $O(td^rF)$.

C.3.6 Hyperparameters

We determine the optimal hyperparameters for our model through systematic exploration. The setting with the best performance on the validation set is selected. The chosen hyperparameters are as follows:

- Number of Hops (r): We set the maximum number of hops to $r = 2$. Empirical evaluation suggests this provides an optimal trade-off between accuracy and computational efficiency.
- Node Signature Dimension (F): The dimension of node signatures, F, is fixed at 1024, except for Citation2 with 512. This configuration ensures that MPLP is both efficient and accurate across all benchmark datasets.

- The minimum degree of nodes to be considered as hubs (b): This parameter indicates the minimum degree of the nodes which are considered as hubs to one-hot encode in the node signatures. We experiment with values in the set $[50, 100, 150]$.
- Batch Size (B): We vary the batch size depending on the graph type: For the 8 non-attributed graphs, we explore batch sizes within $[512, 1024]$. For the 4 attributed graphs coming from [131], we search within $[2048, 4096]$. For OGB datasets, we use 32768 for Collab and PPA, and 261424 for Citation2.

More ablation study can be found in Appendix C.5.4.

C.4 Exploring Bag-Of-Words Node Attributes

In Section 4.4, we delved into the capability of GNNs to discern joint structural features, particularly when presented with Quasi-Orthogonal (QO) vectors. Notably, many graph benchmarks utilize text data to construct node attributes, representing them as Bag-Of-Words (BOW). BOW is a method that counts word occurrences, assigning these counts as dimensional values. With a large dictionary, these BOW node attribute vectors often lean towards QO due to the sparse nature of word representations. Consequently, many node attributes in graph benchmarks inherently possess the QO trait. Acknowledging GNNs' proficiency with QO vector input, we propose the question: *Is it the **QO** property or the **information** embedded within these attributes that significantly impacts link prediction in benchmarks?* This section is an empirical exploration of this inquiry.

C.4.1 Node Attribute Orthogonality

Our inquiry begins with the assessment of node attribute orthogonality across three attributed graphs: CS, Photo, and Collab. CS possesses extensive BOW vocabulary, resulting in node attributes spanning over 8000 dimensions. Contrarily, Photo has a comparatively minimal dictionary, encompassing just 745 dimensions. Collab, deriving node attributes from word embeddings, limits to 128 dimensions.

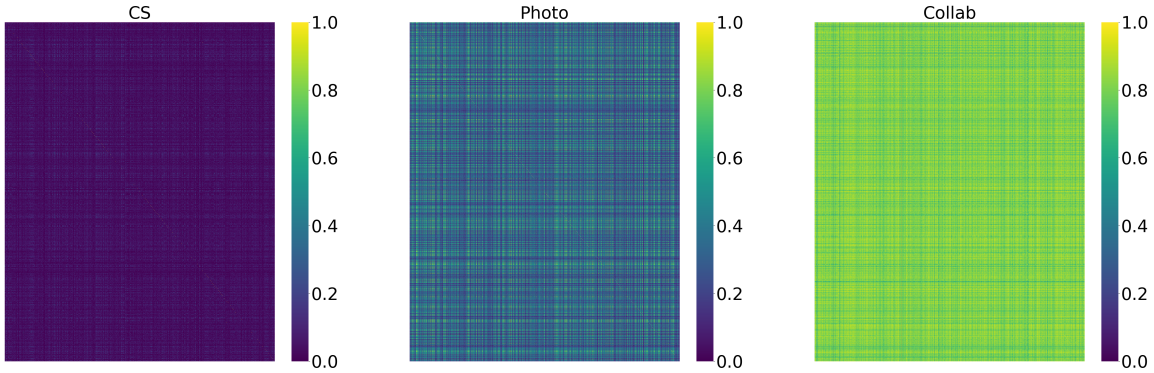

Figure C.2: Heatmap illustrating the inner product of node attributes across CS, Photo, and Collab datasets.

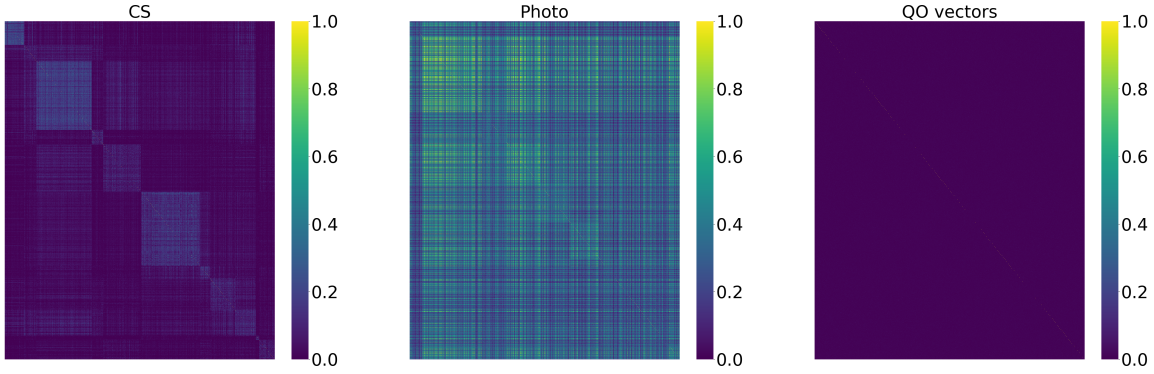

Figure C.3: Heatmap illustrating the inner product of node attributes, arranged by node labels, across CS and Photo. The rightmost showcases the inner product of QO vectors.

For our analysis, we sample 10000 nodes (7650 for Photo) and compute the inner product of their attributes. The results are visualized in Figure C.2. Our findings confirm that with a larger BOW dimension, CS node attributes closely follow QO. However, this orthogonality isn't as pronounced in Photo and Collab—especially Collab, where word embeddings replace BOW. Given that increased node signature dimensions can mitigate estimation variance (as elaborated in Theorem 5), one could posit GNNs might offer enhanced performance on CS, due to its extensive BOW dimensions. Empirical evidence from Table 4.2 supports this claim.

Further, in Figure C.3, we showcase the inner product of node attributes in CS and Photo, but this time, nodes are sequenced by class labels. This order reveals that nodes sharing labels tend to have diminished orthogonality compared to random pairs—a potential variance amplifier in structural feature estimation using node attributes.

C.4.2 Role of Node Attribute Information

To discern the role of embedded information within node attributes, we replace the original attributes in CS, Photo, and Collab with random vectors—denoted as *random feat*. These vectors maintain the original attribute dimensions, though each dimension gets randomly assigned values from $\{-1, 1\}$. The subsequent findings are summarized in Table C.3. Intriguingly, even with this "noise" as input, performance remains largely unaltered. CS attributes appear to convey valuable insights for link predictions, but the same isn't evident for the other datasets. In fact, introducing random vectors to Computers and Photo resulted in enhanced outcomes, perhaps due to their original attribute's insufficient orthogonality hampering effective structural feature capture. Collab shows a performance drop with random vectors, implying that the original word embedding can contribute more to the link prediction than structural feature estimation with merely 128 QO vectors.

TABLE C.3

PERFORMANCE COMPARISON OF GNNS USING NODE ATTRIBUTES VERSUS RANDOM VECTORS (HITS@50).

	CS	**Physics**	**Computers**	**Photo**	**Collab**
GCN	66.00±2.90	73.71±2.28	22.95±10.58	28.14±7.81	35.53±2.39
GCN(random feat)	51.67±2.70	69.55±2.45	35.86±3.17	46.84±2.53	17.25±1.15
SAGE	57.79±18.23	74.10±2.51	1.86±2.53	5.70±10.15	36.82±7.41
SAGE(random feat)	11.78±1.62	64.71±3.65	29.23±3.92	39.94±3.41	28.87±2.36
Random feat					
GCN($F = 1000$)	3.73±1.44	49.28±2.74	36.92±3.36	48.72±3.84	31.93±2.10
GCN($F = 2000$)	24.97±2.67	49.13±4.64	40.24±3.04	53.49±3.50	40.16±1.70
GCN($F = 3000$)	39.51±6.47	53.76±3.85	42.33±3.82	56.27±3.47	47.22±1.60
GCN($F = 4000$)	43.23±3.37	61.86±4.10	42.85±3.60	56.87±3.59	50.40±1.28
GCN($F = 5000$)	48.25±3.28	63.19±4.31	44.52±2.78	58.13±3.79	52.13±1.02
GCN($F = 6000$)	51.44±1.50	65.10±4.11	44.90±2.74	58.10±3.35	53.78±0.84
GCN($F = 7000$)	52.00±1.74	66.76±3.32	45.11±3.69	57.41±2.62	55.04±1.06
GCN($F = 8000$)	54.21±3.47	69.27±2.94	44.47±4.11	58.67±3.90	55.36±1.15
GCN($F = 9000$)	53.16±2.80	70.79±2.83	45.03±3.13	57.15±3.87	OOM
GCN($F = 10000$)	55.91±2.63	71.88±3.29	45.26±1.94	58.12±2.54	OOM

For simplicity, all GNNs are configured with two layers.

C.4.3 Expanding QO Vector Dimensions

Lastly, we substitute node attributes with QO vectors of varied dimensions, utilizing GCN as the encoder. The outcomes of this experiment are cataloged in Table C.3. What's striking is that GCNs, when furnished with lengthier random vectors, often amplify link prediction results across datasets, with the exception of CS. On Computers and Photo, a GCN even rivals our proposed model (Table 4.2), potentially attributed to the enlarged vector dimensions. This suggests that when computational resources permit, expanding our main experiment's node signature dimensions (currently set at 1024) could elevate our model's performance. On Collab. the performance increases significantly compared to the experiments which are input with 128-dimensional vectors, indicating that the structural features are more critical for Collab than the word embedding.

C.5 Additional experiments

C.5.1 Node label estimation accuracy and time

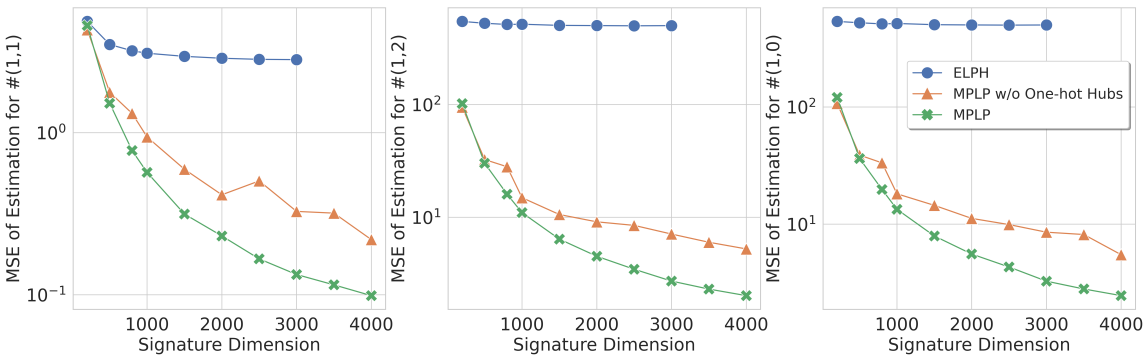

Figure C.4: MSE of estimation for $\#(1,1)$, $\#(1,2)$ and $\#(1,0)$ on Collab. Lower values are better.

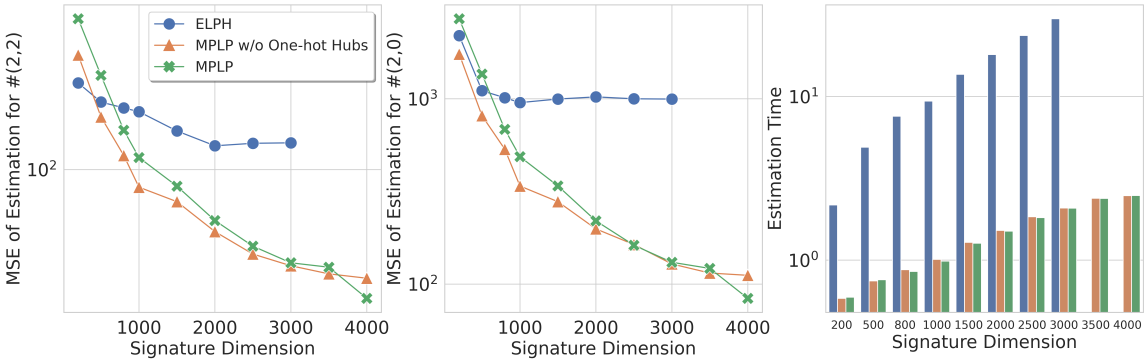

Figure C.5: MSE of estimation for $\#(2,2)$, $\#(2,0)$ and estimation time on Collab. Lower values are better.

In Figure C.4, we assess the accuracy of node label count estimation. For ELPH, the node signature dimension corresponds to the number of MinHash permutations. We employ a default hyperparameter setting for Hyperloglog, with $p = 8$, a configuration that has demonstrated its adequacy in [22]. For time efficiency evaluation, we initially propagate and cache node signatures, followed by performing the estimation.

Furthermore, we evaluate the node label count estimation for $\#(2,2)$ and $\#(2,0)$. The outcomes are detailed in Figure C.5. While MPLP consistently surpasses ELPH in estimation accuracy, the gains achieved via one-hot hubs diminish for $\#(2,2)$ and $\#(2,0)$ relative to node counts at a shortest-path distance of 1. This diminishing performance gain can be attributed to our selection criteria for one-hot encoding, which prioritizes nodes that function as hubs within a one-hop radius. However, one-hop hubs don't necessarily serve as two-hop hubs. While we haven't identified a performance drop for these two-hop node label counts, an intriguing avenue for future research would be to refine variance reduction strategies for both one-hop and two-hop estimations simultaneously.

Regarding the efficiency of estimation, MPLP consistently demonstrates superior computational efficiency in contrast to ELPH. When we increase the node signature dimension to minimize estimation variance, ELPH's time complexity grows exponen-

tially and becomes impractical. In contrast, MPLP displays a sublinear surge in estimation duration.

It's also worth noting that ELPH exhausts available memory when the node signature dimension surpasses 3000. This constraint arises as ELPH, while estimating structural features, has to cache node signatures for both MinHash and Hyperloglog. Conversely, MPLP maintains efficiency by caching only one type of node signatures.

C.5.2 Model enhancement ablation

TABLE C.4

ABLATION STUDY ON NON-ATTRIBUTED BENCHMARKS EVALUATED BY HITS@50.

	USAir	NS	PB	Yeast	C.ele	Power	Router	E.coli
w/o Shortcut removal	80.94±3.49	85.47±2.60	49.51±3.57	82.62±0.99	57.51±2.09	19.99±2.54	36.67±10.03	76.94±1.54
w/o One-hot hubs	**84.04±4.53**	**89.45±2.60**	**51.49±2.63**	**85.11±0.62**	**66.85±3.04**	**29.54±1.79**	**50.81±3.74**	**79.07±2.47**
w/o Norm rescaling	**85.04±2.64**	**89.34±2.79**	**52.50±2.90**	**83.01±1.03**	**66.81±4.11**	**29.00±2.30**	**50.43±3.59**	**79.36±2.18**
MPLP	**85.19±4.59**	**89.58±2.60**	**52.84±3.39**	**85.11±0.62**	**67.97±2.96**	**29.54±1.79**	**51.04±4.03**	**79.35±2.35**

The format is average score ± standard deviation. The top three models are colored by **First**, **Second**, **Third**.

We investigate the individual performance contributions of three primary components in MPLP: Shortcut removal, One-hot hubs, and Norm rescaling. To ensure a fair comparison, we maintain consistent hyperparameters across benchmark datasets, modifying only the specific component under evaluation. Moreover, node attributes are excluded from the model's input for this analysis. The outcomes of this investi-

TABLE C.5

ABLATION STUDY ON ATTRIBUTED BENCHMARKS EVALUATED BY HITS@50.

	CS	Physics	Computers	Photo	Collab
w/o Shortcut removal	41.63±7.27	62.58±2.40	32.74±3.03	52.09±2.52	60.45±1.44
w/o One-hot hubs	**65.49±4.28**	**71.58±2.28**	**36.09±4.08**	**55.63±2.48**	**65.07±0.47**
w/o Norm rescaling	**65.20±2.92**	**67.73±2.54**	**35.83±3.24**	**52.59±3.57**	**63.99±0.59**
MPLP	**65.70±3.86**	**71.03±3.55**	**37.56±3.57**	**55.63±2.48**	**66.07±0.47**

The format is average score ± standard deviation. The top three models are colored by **First**, **Second**, **Third**.

gation are detailed in Table C.4 and Table C.5.

Among the three components, Shortcut removal emerges as the most pivotal for MPLP. This highlights the essential role of ensuring the structural distribution of positive links is aligned between the training and testing datasets [37].

Regarding One-hot hubs, while they exhibited strong results in the estimation accuracy evaluations presented in Figure C.4 and Figure C.5, their impact on the overall performance is relatively subdued. We hypothesize that, in the context of these sparse benchmark graphs, the estimation variance may not be sufficiently influential on the model's outcomes.

Finally, Norm rescaling stands out as a significant enhancement in MPLP. This is particularly evident in its positive impact on datasets like Yeast, Physics, Photo, and Collab.

C.5.3 Structural features ablation

We further examine the contribution of various structural features to the link prediction task. These features include: $\#(1,1)$, $\#(1,2)$, $\#(1,0)$, $\#(2,2)$, $\#(2,0)$, and $\#(\triangle)$. To ensure fair comparison, we utilize only the structural features for link representation, excluding the node representations derived from GNN$(\cdot)$. Given the combinatorial nature of these features, they are grouped into four categories:

- $\#(1,1)$;
- $\#(1,2)$, $\#(1,0)$;
- $\#(2,2)$, $\#(2,0)$;
- $\#(\triangle)$.

The configuration of these structural features and their corresponding results are detailed in Table C.6 and Table C.7.

Our analysis reveals that distinct benchmark datasets have varied preferences for structural features, reflecting their unique underlying distributions. For example, datasets PB and Power exhibit superior performance with 2-hop structural features, whereas others predominantly favor 1-hop features. Although $\#(1,1)$, which counts Common Neighbors, is often considered pivotal for link prediction, the two other 1-hop structural features, $\#(1,2)$ and $\#(1,0)$, demonstrate a more pronounced impact on link prediction outcomes. Meanwhile, while the count of triangles, $\#(\triangle)$, possesses theoretical significance for model expressiveness, it seems less influential for link prediction when assessed in isolation. However, its presence can bolster link prediction performance when combined with other key structural features.

C.5.4 Parameter sensitivity

We perform an ablation study to assess the hyperparameter sensitivity of MPLP, focusing specifically on two parameters: Batch Size (B) and Node Signature Dimen-

sion (F).

Our heightened attention to B stems from its role during training. Within each batch, MPLP executes the shortcut removal. Ideally, if $B = 1$, only one target link would be removed, thereby preserving the local structures of other links. However, this approach is computationally inefficient. Although shortcut removal can markedly enhance performance and address the distribution shift issue (as elaborated in Appendix C.5.2), it can also inadvertently modify the graph structure. Thus, striking a balance between computational efficiency and minimal graph structure alteration is essential.

Our findings are delineated in Table C.8, Table C.9, Table C.10, and Table C.11. Concerning the batch size, our results indicate that opting for a smaller batch size typically benefits performance. However, if this size is increased past a certain benchmark threshold, there can be a noticeable performance drop. This underscores the importance of pinpointing an optimal batch size for MPLP. Regarding the node signature dimension, our data suggests that utilizing longer QO vectors consistently improves accuracy by reducing variance. This implies that, where resources allow, selecting a more substantial node signature dimension is consistently advantageous.

C.6 Theoretical analysis

C.6.1 Proof for Theorem 4

We begin by restating Theorem 4 and then proceed with its proof:

Let $G = (V, E)$ be a non-attributed graph and consider a 1-layer GCN/SAGE. Define the input vectors $\boldsymbol{X} \in \mathbb{R}^{N \times F}$ initialized randomly from a zero-mean distribution with standard deviation σ_{node}. Additionally, let the weight matrix $\boldsymbol{W} \in \mathbb{R}^{F' \times F}$ be initialized from a zero-mean distribution with standard deviation σ_{weight}. After performing message passing, for any pair of nodes $\{(u, v) | (u, v) \in V \times V \setminus E\}$, the

expected value of their inner product is given by:

For GCN:

$$\mathbb{E}(\boldsymbol{h}_u \cdot \boldsymbol{h}_v) = \frac{C}{\sqrt{\hat{d}_u \hat{d}_v}} \sum_{k \in \mathcal{N}_u \bigcap \mathcal{N}_v} \frac{1}{\hat{d}_k},$$

For SAGE:

$$\mathbb{E}(\boldsymbol{h}_u \cdot \boldsymbol{h}_v) = \frac{C}{\sqrt{d_u d_v}} \sum_{k \in \mathcal{N}_u \bigcap \mathcal{N}_v} 1,$$

where $\hat{d}_v = d_v + 1$ and the constant C is defined as $C = \sigma_{node}^2 \sigma_{weight}^2 FF'$.

Proof. Define $\boldsymbol{X}$ as $\left(\boldsymbol{X}_1^\top, \ldots, \boldsymbol{X}_N^\top\right)^\top$ and $\boldsymbol{W}$ as $(\boldsymbol{W}_1, \boldsymbol{W}_2, \ldots, \boldsymbol{W}_F)$.

Using GCN as the MPNN, the node representation is updated by:

$$\boldsymbol{h}_u = \boldsymbol{W} \sum_{k \in \mathcal{N}(u) \cup \{u\}} \frac{1}{\sqrt{\hat{d}_k \hat{d}_u}} \boldsymbol{X}_k,$$

where $\hat{d}_v = d_v + 1$.

For any two nodes (u, v) from $\{(u, v) | (u, v) \in V \times V \setminus E\}$, we compute:

$$\begin{aligned}
\boldsymbol{h}_u \cdot \boldsymbol{h}_v &= \boldsymbol{h}_u^\top \boldsymbol{h}_v \\
&= \left(\boldsymbol{W} \sum_{a \in \mathcal{N}(u) \cup \{u\}} \frac{1}{\sqrt{\hat{d}_a \hat{d}_u}} \boldsymbol{X}_a \right)^\top \left(\boldsymbol{W} \sum_{b \in \mathcal{N}(v) \cup \{v\}} \frac{1}{\sqrt{\hat{d}_b \hat{d}_v}} \boldsymbol{X}_b \right) \\
&= \sum_{a \in \mathcal{N}(u) \cup \{u\}} \frac{1}{\sqrt{\hat{d}_a \hat{d}_u}} \boldsymbol{X}_a^\top \boldsymbol{W}^\top \boldsymbol{W} \sum_{b \in \mathcal{N}(v) \cup \{v\}} \frac{1}{\sqrt{\hat{d}_b \hat{d}_v}} \boldsymbol{X}_b \\
&= \sum_{a \in \mathcal{N}(u) \cup \{u\}} \frac{1}{\sqrt{\hat{d}_a \hat{d}_u}} \boldsymbol{X}_a^\top \begin{pmatrix} \boldsymbol{W}_1^\top \boldsymbol{W}_1 & \cdots & \boldsymbol{W}_1^\top \boldsymbol{W}_F \\ \vdots & \vdots & \vdots \\ \boldsymbol{W}_F^\top \boldsymbol{W}_1 & \cdots & \boldsymbol{W}_F^\top \boldsymbol{W}_F \end{pmatrix} \sum_{b \in \mathcal{N}(v) \cup \{v\}} \frac{1}{\sqrt{\hat{d}_b \hat{d}_v}} \boldsymbol{X}_b.
\end{aligned}$$

Given that

1. $\mathbb{E}\big(\boldsymbol{W}_i^\top \boldsymbol{W}_j\big) = \sigma^2_{weight} F'$ when $i = j$,
2. $\mathbb{E}\big(\boldsymbol{W}_i^\top \boldsymbol{W}_j\big) = 0$ when $i \neq j$,

we obtain:

$$\mathbb{E}(\boldsymbol{h}_u \cdot \boldsymbol{h}_v) = \sigma^2_{weight} F' \sum_{a \in \mathcal{N}(u) \cup \{u\}} \frac{1}{\sqrt{\hat{d}_a \hat{d}_u}} \boldsymbol{X}_a^\top \sum_{b \in \mathcal{N}(v) \cup \{v\}} \frac{1}{\sqrt{\hat{d}_b \hat{d}_v}} \boldsymbol{X}_b.$$

Also the orthogonal of the random vectors guarantee that $\mathbb{E}\big(\boldsymbol{X}_a^\top \boldsymbol{X}_b\big) = 0$ when $a \neq b$. Then, we have:

$$\mathbb{E}(\boldsymbol{h}_u \cdot \boldsymbol{h}_v) = \frac{C}{\sqrt{\hat{d}_u \hat{d}_v}} \sum_{k \in \mathcal{N}_u \bigcap \mathcal{N}_v} \frac{1}{\hat{d}_k}$$

where $C = \sigma^2_{node} \sigma^2_{weight} F F'$.

This completes the proof for the GCN variant. A similar approach, utilizing the probabilistic orthogonality of the input vectors and weight matrix, can be employed to derive the expected value for SAGE as the MPNN. □

C.6.2 Proof for Theorem 5

We begin by restating Theorem 5 and then proceed with its proof:

Let $G = (V, E)$ be a graph, and let the vector dimension be given by $F \in \mathbb{N}_+$. Define the input vectors $\boldsymbol{X} = (X_{i,j})$, which are initialized from a random variable x having a mean of 0 and a standard deviation of $\frac{1}{\sqrt{F}}$. Using the message-passing as described by Equation 4.3, for any pair of nodes $\{(u, v) | (u, v) \in V \times V\}$, the expected value and variance of their inner product are:

$$\begin{aligned} \mathbb{E}(\boldsymbol{h}_u \cdot \boldsymbol{h}_v) &= \mathrm{CN}(u, v), \\ \mathrm{Var}(\boldsymbol{h}_u \cdot \boldsymbol{h}_v) &= \frac{1}{F} \big(d_u d_v + \mathrm{CN}(u, v)^2 - 2\mathrm{CN}(u, v)\big) + F \mathrm{Var}\big(\mathrm{x}^2\big) \mathrm{CN}(u, v). \end{aligned}$$

Proof. We follow the proof of the theorem in [114]. Based on the message-passing defined in Equation 4.3:

$$\begin{aligned}
\mathbb{E}(\boldsymbol{h}_u \cdot \boldsymbol{h}_v) &= \mathbb{E}\left(\left(\sum_{k_u \in \mathcal{N}_u} \boldsymbol{X}_{k_u,:}\right) \cdot \left(\sum_{k_v \in \mathcal{N}_v} \boldsymbol{X}_{k_v,:}\right)\right) \\
&= \mathbb{E}\left(\sum_{k_u \in \mathcal{N}_u} \sum_{k_v \in \mathcal{N}_v} \boldsymbol{X}_{k_u,:} \boldsymbol{X}_{k_v,:}\right) \\
&= \sum_{k_u \in \mathcal{N}_u} \sum_{k_v \in \mathcal{N}_v} \mathbb{E}(\boldsymbol{X}_{k_u,:} \boldsymbol{X}_{k_v,:}).
\end{aligned}$$

Since the sampling of each dimension is independent of each other, we get:

$$\mathbb{E}(\boldsymbol{h}_u \cdot \boldsymbol{h}_v) = \sum_{k_u \in \mathcal{N}_u} \sum_{k_v \in \mathcal{N}_v} \sum_{i=1}^{F} \mathbb{E}(X_{k_u,i} X_{k_v,i}).$$

When $k_u = k_v$,

$$\mathbb{E}(X_{k_u,i} X_{k_v,i}) = \mathbb{E}\left(\mathrm{x}^2\right) = \frac{1}{F}.$$

When $k_u \neq k_v$,

$$\mathbb{E}(X_{k_u,i} X_{k_v,i}) = \mathbb{E}(X_{k_u,i}) \mathbb{E}(X_{k_v,i}) = 0.$$

Thus:

$$\begin{aligned}
\mathbb{E}(\boldsymbol{h}_u \cdot \boldsymbol{h}_v) &= \sum_{k_u \in \mathcal{N}_u} \sum_{k_v \in \mathcal{N}_v} \sum_{i=1}^{F} \mathbf{1}\left(k_u = k_v\right) \frac{1}{F} \\
&= \sum_{k \in \mathcal{N}_u \cap \mathcal{N}_v} 1 = \mathrm{CN}(u, v).
\end{aligned}$$

For the variance, we separate the equal from the non-equal pairs of k_u and k_v. Note that there is no covariance between the equal pairs and the non-equal pairs due

to the independence:

$$\begin{aligned}\mathrm{Var}(\boldsymbol{h}_u \cdot \boldsymbol{h}_v) &= \mathrm{Var}\left(\sum_{k_u \in \mathcal{N}_u} \sum_{k_v \in \mathcal{N}_v} \sum_{i=1}^{F} X_{k_u,i} X_{k_v,i}\right) \\ &= \sum_{i=1}^{F} \mathrm{Var}\left(\sum_{k_u \in \mathcal{N}_u} \sum_{k_v \in \mathcal{N}_v} X_{k_u,i} X_{k_v,i}\right) \\ &= \sum_{i=1}^{F} \left(\mathrm{Var}\left(\sum_{k \in \mathcal{N}_u \cap \mathcal{N}_v} \mathrm{x}^2\right) + \mathrm{Var}\left(\sum_{k_u \in \mathcal{N}_u} \sum_{k_v \in \mathcal{N}_v \setminus \{k_u\}} X_{k_u,i} X_{k_v,i}\right)\right).\end{aligned}$$

For the first term, we can obtain:

$$\mathrm{Var}\left(\sum_{k \in \mathcal{N}_u \cap \mathcal{N}_v} \mathrm{x}^2\right) = \mathrm{Var}\big(\mathrm{x}^2\big)\mathrm{CN}(u, v).$$

For the second term, we further split the variance of linear combinations to the linear combinations of variances and covariances:

$$\begin{aligned}\mathrm{Var}\left(\sum_{k_u \in \mathcal{N}_u} \sum_{k_v \in \mathcal{N}_v \setminus \{k_u\}} X_{k_u,i} X_{k_v,i}\right) = &\sum_{k_u \in \mathcal{N}_u} \sum_{k_v \in \mathcal{N}_v \setminus \{k_u\}} \mathrm{Var}(X_{k_u,i} X_{k_v,i}) + \\ &\sum_{a \in \mathcal{N}_u \setminus \{k_u\}} \sum_{b \in \mathcal{N}_v \setminus \{k_v, a\}} \mathrm{Cov}(X_{k_u,i} X_{k_v,i}, X_{a,i} X_{b,i}).\end{aligned}$$

Note that the $\mathrm{Cov}(X_{k_u,i} X_{k_v,i}, X_{a,i} X_{b,i})$ is $\mathrm{Var}(X_{k_u,i} X_{k_v,i}) = \frac{1}{F^2}$ when $(k_u, k_v) = (b, a)$, and otherwise 0.

Thus, we have:

$$\mathrm{Var}\left(\sum_{k_u \in \mathcal{N}_u} \sum_{k_v \in \mathcal{N}_v \setminus \{k_u\}} X_{k_u,i} X_{k_v,i}\right) = \frac{1}{F^2}\left(d_u d_v + \mathrm{CN}(u, v)^2 - 2\mathrm{CN}(u, v)\right),$$

and the variance is:

$$\mathrm{Var}(\boldsymbol{h}_u \cdot \boldsymbol{h}_v) = \frac{1}{F}\left(d_u d_v + \mathrm{CN}(u, v)^2 - 2\mathrm{CN}(u, v)\right) + F\mathrm{Var}\big(\mathrm{x}^2\big)\mathrm{CN}(u, v).$$

□

C.6.3 Proof for Theorem 6

We begin by restating Theorem 6 and then proceed with its proof:

Under the conditions defined in Theorem 5, let $\boldsymbol{h}_u^{(l)}$ denote the vector for node u after the l-th message-passing iteration. We have:

$$\mathbb{E}\big(\boldsymbol{h}_u^{(p)} \cdot \boldsymbol{h}_v^{(q)}\big) = \sum_{k \in V} |\text{walks}^{(p)}(k, u)||\text{walks}^{(q)}(k, v)|,$$

where $|\text{walks}^{(l)}(u, v)|$ counts the number of length-l walks between nodes u and v.

Proof. Reinterpreting the message-passing described in Equation 4.3, we can equivalently express it as:

$$\text{ms}_v^{(l+1)} = \bigcup_{u \in \mathcal{N}_v} \text{ms}_u^{(l)}, \boldsymbol{h}_v^{(l+1)} = \sum_{u \in \text{ms}_v^{(l+1)}} \boldsymbol{h}_u^{(0)}, \tag{C.2}$$

where $\text{ms}_v^{(l)}$ refers to a multiset, a union of multisets from its neighbors. Initially, $\text{ms}_v^{(0)} = \{\{v\}\}$. The node vector $\boldsymbol{h}_v^{(l)}$ is derived by summing the initial QO vectors of the multiset's elements.

We proceed by induction: Base Case ($l = 1$):

$$\text{ms}_v^{(1)} = \bigcup_{u \in \mathcal{N}_v} \text{ms}_u^{(0)} = \bigcup_{u \in \mathcal{N}_v} \{\{u\}\} = \{\{k | \omega \in \text{walks}^{(1)}(k, v)\}\}$$

Inductive Step ($l \geq 1$): Let's assume that $\text{ms}_v^{(l)} = \{\{k | \omega \in \text{walks}^{(l)}(k, v)\}\}$ holds true for an arbitrary l. Utilizing Equation C.2 and the inductive hypothesis, we deduce:

$$\text{ms}_v^{(l+1)} = \bigcup_{u \in \mathcal{N}_v} \{\{k | \omega \in \text{walks}^{(l)}(k, u)\}\}.$$

If k initiates the l-length walks terminating at v and if v is adjacent to u, then k must similarly initiate the l-length walks terminating at u. This consolidates our inductive premise.

With the induction established:

$$\mathbb{E}\big(\boldsymbol{h}_u^{(p)} \cdot \boldsymbol{h}_v^{(q)}\big) = \mathbb{E}\left(\sum_{k_u \in \mathrm{ms}_u^{(p)}} \boldsymbol{h}_{k_u}^{(0)} \cdot \sum_{k_v \in \mathrm{ms}_v^{(q)}} \boldsymbol{h}_{k_v}^{(0)}\right)$$

The inherent independence among node vectors concludes the proof. □

C.7 Limitations

Despite the promising capabilities of MPLP, there are distinct limitations that warrant attention:

1. Training cost vs. inference cost: The computational cost during training significantly outweighs that of inference. This arises from the necessity to remove shortcut edges for positive links in the training phase, causing the graph structure to change across different batches. This, in turn, mandates a repeated computation of the shortest-path neighborhood. Even though MPLP+ can avoid the computation of the shortest-path neighborhood for each batch, it shows suboptimal performance compared to MPLP. A potential remedy is to consider only a subset of links in the graph as positive instances and mask them, enabling a single round of preprocessing. Exploring this approach will be the focus of future work.

2. Estimation variance influenced by graph structure: The structure of the graph itself can magnify the variance of our estimations. Specifically, in dense graphs or those with a high concentration of hubs, the variance can become substantial, thereby compromising the accuracy of structural feature estimation.

3. Optimality of estimating structural features: Our research demonstrates the feasibility of using message-passing to derive structural features. However, its optimality remains undetermined. Message-passing, by nature, involves sparse matrix multiplication operations, which can pose challenges in terms of computational time and space, particularly for exceedingly large graphs.

TABLE C.6

THE MAPPING BETWEEN THE CONFIGURATION NUMBER AND THE USED STRUCTURAL FEATURES IN MPLP.

Configurations	$\#(1,1)$	$\#(1,2)$	$\#(1,0)$	$\#(2,2)$	$\#(2,0)$	$\#(\triangle)$
(1)	✓	-	-	-	-	-
(2)	-	✓	✓	-	-	-
(3)	-	-	-	✓	✓	-
(4)	-	-	-	-	-	✓
(5)	✓	✓	✓	-	-	-
(6)	✓	-	-	✓	✓	-
(7)	✓	-	-	-	-	✓
(8)	-	✓	✓	✓	✓	-
(9)	-	✓	✓	-	-	✓
(10)	-	-	-	✓	✓	✓
(11)	✓	✓	✓	✓	✓	-
(12)	✓	✓	✓	-	-	✓
(13)	✓	-	-	✓	✓	✓
(14)	-	✓	✓	✓	✓	✓
(15)	✓	✓	✓	✓	✓	✓

TABLE C.7

ABLATION ANALYSIS ON STRUCTURAL FEATURES.

Configurations	**USAir**	**NS**	**PB**	**Yeast**	**C.ele**	**Power**	**Router**	**E.coli**
(1)	76.64±26.74	75.26±2.79	37.48±13.30	58.70±30.50	46.22±24.84	14.40±1.40	17.29±3.96	60.10±30.80
(2)	82.54±4.61	84.76±3.63	41.84±15.51	80.56±0.65	56.22±20.39	21.38±1.46	48.97±3.34	67.78±23.83
(3)	67.76±23.65	70.05±2.35	44.81±2.63	67.02±2.53	36.53±19.68	25.24±4.07	21.32±2.66	56.59±1.78
(4)	37.18±37.57	25.13±1.99	12.35±10.75	7.42±10.80	30.75±18.69	5.47±1.13	30.47±3.10	34.90±36.63
(5)	86.24±2.70	84.91±2.80	48.35±3.76	84.42±0.56	66.69±3.60	22.25±1.39	49.68±3.79	80.94±1.62
(6)	77.41±5.27	80.00±2.39	46.05±2.76	74.70±1.45	46.88±5.79	27.74±3.23	22.37±2.06	71.41±2.47
(7)	71.11±25.51	76.72±2.37	43.57±3.70	73.08±1.23	54.99±20.14	14.50±1.64	31.26±2.87	80.22±2.09
(8)	80.16±4.82	88.67±2.72	52.16±2.25	82.52±0.85	63.82±4.02	28.41±2.00	50.97±3.57	77.26±1.31
(9)	75.13±26.51	87.28±3.33	48.10±3.43	80.84±0.97	60.63±4.54	23.85±1.37	49.78±3.56	76.13±1.81
(10)	76.82±4.28	77.04±3.70	45.42±2.77	67.34±3.20	41.66±13.47	26.95±1.47	28.31±2.76	70.14±0.77
(11)	82.82±5.52	88.91±2.90	52.57±3.05	84.61±0.67	67.11±2.52	28.98±1.73	50.63±3.72	80.16±2.20
(12)	87.29±1.08	88.08±2.59	48.86±3.42	84.59±0.69	66.06±3.74	23.79±1.87	50.06±3.66	79.57±2.46
(13)	78.21±2.74	88.08±3.27	46.00±2.31	74.88±2.49	54.64±4.99	28.82±1.29	26.24±2.18	74.67±3.96
(14)	80.75±5.02	89.14±2.38	51.63±2.67	82.68±0.67	63.01±3.21	29.41±1.44	51.08±4.12	76.88±1.86
(15)	81.06±6.62	89.73±2.12	53.49±2.66	85.06±0.69	66.41±3.02	28.86±2.40	50.63±3.79	78.91±2.58

Ablation analysis highlighting the impact of various structural features on link prediction. Refer to Table C.6 for detailed configurations of the structural features used.

TABLE C.8

ABLATION STUDY OF BATCH SIZE (B) ON NON-ATTRIBUTED BENCHMARKS EVALUATED BY HITS@50.

	USAir	NS	PB	Yeast	C.ele	Power	Router	E.coli
MPLP($B = 256$)	**90.31**$\pm$1.32	**88.98**$\pm$2.48	**51.14**$\pm$2.44	**84.07**$\pm$0.69	**71.59**$\pm$2.83	**28.92**$\pm$1.67	**56.15**$\pm$3.80	**85.12**$\pm$1.00
MPLP($B = 512$)	**90.40**$\pm$2.47	**89.40**$\pm$2.12	49.63$\pm$2.08	**84.17**$\pm$0.60	**71.72**$\pm$3.35	**28.60**$\pm$1.66	**53.25**$\pm$6.57	84.72$\pm$1.04
MPLP($B = 1024$)	**90.49**$\pm$2.22	**88.49**$\pm$2.34	50.60$\pm$3.40	**83.67**$\pm$0.57	**70.61**$\pm$4.13	**28.63**$\pm$1.60	49.75$\pm$5.14	84.52$\pm$1.03
MPLP($B = 2048$)	81.20$\pm$2.80	61.79$\pm$18.55	50.34$\pm$3.05	76.79$\pm$6.79	31.79$\pm$19.88	28.45$\pm$1.88	49.37$\pm$3.89	84.43$\pm$1.28
MPLP($B = 4096$)	81.20$\pm$2.80	61.79$\pm$18.55	**52.59**$\pm$2.36	58.26$\pm$7.20	31.54$\pm$18.53	27.25$\pm$3.30	**50.26**$\pm$3.89	**85.15**$\pm$1.15
MPLP($B = 8192$)	81.20$\pm$2.80	56.20$\pm$21.34	**51.91**$\pm$2.08	24.47$\pm$21.12	31.79$\pm$19.88	17.22$\pm$3.17	38.67$\pm$7.78	**85.67**$\pm$0.90

The format is average score $\pm$ standard deviation. The top three models are colored by **First**, **Second**, **Third**.

TABLE C.9

ABLATION STUDY OF BATCH SIZE (B) ON ATTRIBUTED BENCHMARKS EVALUATED BY HITS@50.

	CS	Physics	Computers	Photo
MPLP($B = 256$)	74.96$\pm$1.87	**76.06**$\pm$1.47	**43.38**$\pm$2.83	**57.58**$\pm$2.92
MPLP($B = 512$)	**75.61**$\pm$2.25	**75.38**$\pm$1.79	**42.95**$\pm$2.56	**57.19**$\pm$2.51
MPLP($B = 1024$)	74.89$\pm$2.00	74.89$\pm$1.97	**42.69**$\pm$2.41	**56.97**$\pm$3.20
MPLP($B = 2048$)	75.02$\pm$2.68	**75.47**$\pm$1.68	41.39$\pm$2.87	55.89$\pm$3.03
MPLP($B = 4096$)	**75.46**$\pm$1.78	74.88$\pm$2.57	40.65$\pm$2.85	55.89$\pm$2.88
MPLP($B = 8192$)	**75.26**$\pm$1.91	74.14$\pm$2.17	40.00$\pm$3.40	55.90$\pm$2.52

The format is average score $\pm$ standard deviation. The top three models are colored by **First**, **Second**, **Third**.

TABLE C.10

ABLATION STUDY OF NODE SIGNATURE DIMENSION (F) ON NON-ATTRIBUTED BENCHMARKS EVALUATED BY HITS@50.

	USAir	**NS**	**PB**	**Yeast**	**C.ele**	**Power**	**Router**	**E.coli**
MPLP($F = 256$)	**90.64**±2.50	88.52±3.07	50.42±3.86	80.63±0.84	70.89±4.70	25.74±1.59	51.84±2.90	84.60±0.92
MPLP($F = 512$)	**90.49**±1.95	89.18±2.35	**51.48**±2.63	82.41±1.10	**70.91**±4.68	27.58±1.80	51.98±4.38	**84.70**±1.33
MPLP($F = 1024$)	**90.16**±1.61	**89.40**±2.12	50.60±3.40	**83.87**±1.06	70.61±4.13	**28.88**±2.24	**53.92**±2.88	**84.81**±0.85
MPLP($F = 2048$)	90.14±2.24	**89.36**±1.92	**51.26**±1.67	**84.20**±1.02	**72.24**±3.31	**29.27**±1.92	**54.50**±4.52	84.58±1.42
MPLP($F = 4096$)	89.95±1.48	**89.54**±2.22	**51.07**±2.87	**84.89**±0.64	**71.91**±3.52	**29.26**±1.51	**54.71**±5.07	**84.67**±0.61

The format is average score ± standard deviation. The top three models are colored by **First**, **Second**, **Third**.

TABLE C.11

ABLATION STUDY OF NODE SIGNATURE DIMENSION (F) ON ATTRIBUTED BENCHMARKS EVALUATED BY HITS@50.

	CS	**Physics**	**Computers**	**Photo**
MPLP($F = 256$)	74.90±1.88	73.91±1.41	40.65±3.24	55.13±2.98
MPLP($F = 512$)	74.67±2.63	74.49±2.05	39.36±2.28	**55.93**±3.31
MPLP($F = 1024$)	**75.02**±2.68	**75.27**±2.95	**42.27**±3.96	55.89±3.03
MPLP($F = 2048$)	**75.30**±2.14	**75.82**±2.15	**41.98**±3.21	**57.11**±2.56
MPLP($F = 4096$)	**76.04**±1.57	**76.17**±2.04	**43.33**±2.93	**58.55**±2.47

The format is average score ± standard deviation. The top three models are colored by **First**, **Second**, **Third**.

APPENDIX D

YOU DO NOT HAVE TO TRAIN GRAPH NEURAL NETWORKS AT ALL ON TEXT-ATTRIBUTED GRAPHS

D.1 More strategical design

While the weight matrix obtained by Equation 5.8 is sufficient for a predictive TrainlessGNN, we further enhance the performance by introducing two more strategical design for our method.

Utilizing Propagated Node Attributes. In updating the representation of virtual label nodes during message passing in Equation 5.8, nodes in the training set are initially associated with their original node attributes $\mathbf{X}_{\mathcal{L}}$. An alternative approach is to initialize the labeled nodes with a smoothed version of the node attributes. To execute this, prior to integrating the virtual label nodes into the graph, we conduct l rounds of message passing as defined by $\mathrm{AGG}(\cdot)$. This step refines each labeled node's representation over the graph structure. Post this refinement, we introduce the virtual label nodes into the graph and proceed as earlier, but now associating labeled nodes with the propagated node attributes. Specifically, the weight matrix $\mathbf{W}$ is now obtained from Equation 5.8 as:

$$\mathbf{W}^{\top} = (\mathbf{B}_{\mathcal{L}} - \frac{\omega}{C}\mathbf{1})^{\top}\mathbf{H}_{\mathcal{L}}^{(l)}, \tag{D.1}$$

where $\mathbf{H}_{\mathcal{L}}^{(l)}$ denotes the smoothed node representation of labeled nodes. This adjustment is beneficial when the inner product within the same class nodes is not

significant. The propagated node attributes can help keep the node attributes closer to the corresponding class's trainless weight vector. It's worth noting that any GNN can implement the message passing function AGG(·). In practice, the message passing settings of GCN/SGC are chosen for use.

Weighted message passing. The method used to obtain the weight matrix by propagating node attributes from the same class as in Equation D.1 treats each node equally, without considering their local structural information. To integrate this structural information into the weight matrix calculation, we introduce a weighted message passing technique. This approach is inspired by the Adamic Adar index (AA) [4] and Resource Allocation (RA) [191], utilized in link prediction. In this context, messages from nodes with fewer neighbors are given more weight compared to messages from hub nodes. On the contrary, the unweighted version can be seen as an approach akin to Common Neighbor (CN) [91]. The formal expression for message passing is then reformulated as:

$$\mathbf{W}^\top = \mathbf{R}_\mathcal{L}(\mathbf{B}_\mathcal{L} - \frac{\omega}{C}\mathbf{1})^\top \mathbf{H}_\mathcal{L}^{(l)}, \tag{D.2}$$

where $\mathbf{R}_\mathcal{L}$ is the degree normalization matrix specific to the labeled nodes. To compute $\mathbf{R}_\mathcal{L}$, we initially determine the normalization matrix $\mathbf{R}$ for the entire node set, which encompasses both $\mathcal{L}$ and $\mathcal{U}$. Based on the degree matrix $\mathbf{D}$, for AA, we have $\mathbf{R} = (\log \mathbf{D})^{-1}$ and for RA, $\mathbf{R} = \mathbf{D}^{-1}$. For the CN-type message passing, we simply set $\mathbf{R} = \mathbf{I}$. $\mathbf{R}_\mathcal{L}$ is then acquired by extracting the corresponding portions from $\mathbf{R}$ for labeled nodes.

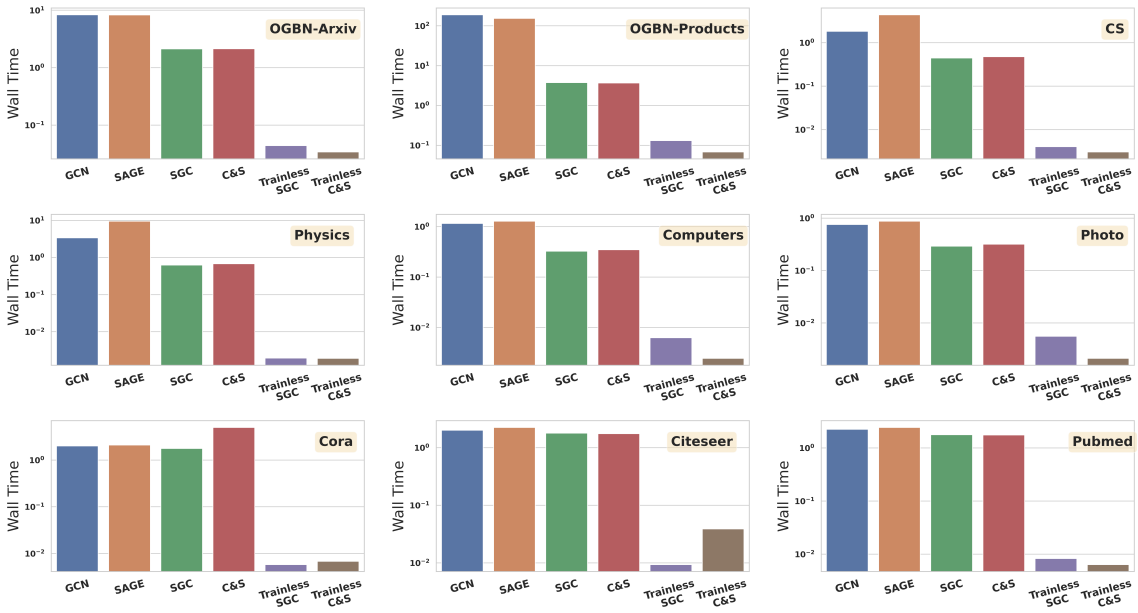

Figure D.1: Training efficiency of TrainlessGNN compared to the traditional gradient descent optimizations.

D.2 Supplementary experiments

D.2.1 Statistics of benchmark datasets

We show the statistics of the benchmark datasets in Table D.1. The data reveals that the three Planetoid datasets (Cora, Citeseer, and Pubmed) along with the two Coauthor datasets (CS and Physics) exhibit clear characteristics of over-parameterization and quasi-orthogonality. These traits contribute to TrainlessGNN achieving superior performance even when compared to trained models. Conversely, the Amazon co-purchase networks, despite showing over-parameterization, exhibit weaker quasi-orthogonality (See Figure 5.1). For the other two OGB datasets (OGBN-Arxiv and OGBN-Products), the abundance of labeled nodes in the training sets alleviates the semi-supervised node classification task for the trained models.

TABLE D.1

THE STATISTICS OF THE BENCHMARK DATASETS.

Dataset	#Nodes	#Edges	#Classes	Attr. Dimension	Text Encoding	#Training/#Validation/#Testing
Cora	2708	10556	7	1433	BOW	140/500/1000
Citeseer	3327	9104	6	3703	BOW	120/500/1000
Pubmed	19717	88648	3	500	TF-IDF	60/500/1000
CS	18333	163788	15	6805	BOW	300/450/17583
Physics	34493	495924	5	8415	BOW	100/150/34243
Computers	13752	491722	10	767	BOW	200/300/13252
Photo	7650	238162	8	745	BOW	160/240/7250
OGBN-Products	2449029	123718152	47	100	BOW+PCA	196615/39323/2213091
OGBN-Arxiv	169343	2315598	40	128	Average of word embedding	90941/29799/48603

D.2.2 Baseline model details

Baseline method C&S. Recall that the C&S approach [64] commences training with an MLP solely on node attributes, disregarding the graph structure. It then propagates the computed logits through the graph to refine predictions based on the graph structure, as formulated:

$$\hat{\mathbf{Z}} = \mathrm{MLP}(\mathbf{X}), \mathbf{Z} = \mathrm{C\&S}(\hat{\mathbf{Z}}, \mathbf{A}), \tag{D.3}$$

where $\hat{\mathbf{Z}}$ represents the logits for the node classification tasks. The logit propagation process in C&S$(\cdot)$ encompasses a two-step sequence, namely the Correct$(\cdot)$ step and Smooth$(\cdot)$ step:

$$\mathrm{C\&S} = \mathrm{Smooth} \circ \mathrm{Correct}(\hat{\mathbf{Z}}, \mathbf{A}, \mathbf{B}), \tag{D.4}$$

where $\mathbf{B}$ denotes the one-hot encoding of the labels.

The Correct(·) step is implemented as:

$$\mathbf{E}^{(\ell)} = \alpha_1 \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\mathbf{E}^{(\ell-1)} + (1-\alpha_1)\mathbf{E}^{(\ell-1)} \tag{D.5}$$

$$\mathbf{Z}' = \mathbf{Z} + \gamma \cdot \mathbf{E}^{(L_1)}, \tag{D.6}$$

where $\mathbf{E}^{(\ell)} \in \mathbb{R}^{n\times C}$ symbolizes the error matrix and γ represents the scaling factor. The error matrix is defined as follows:

$$\mathbf{E}_i^{(0)} = \begin{cases} \mathbf{B}_i - \hat{\mathbf{Z}}_i, & \text{if } v_i \in \mathcal{L}, \\ \mathbf{0}, & \text{else.} \end{cases} \tag{D.7}$$

Subsequently, the Correct(·) step is defined as:

$$\mathbf{Z}_i^{(0)} = \begin{cases} \mathbf{B}_i, & \text{if } v_i \in \mathcal{L}, \\ \mathbf{Z}'_i, & \text{else} \end{cases} \tag{D.8}$$

$$\mathbf{Z}^{(\ell)} = \alpha_2 \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\mathbf{Z}^{(\ell-1)} + (1-\alpha_2)\mathbf{Z}^{(\ell-1)} \tag{D.9}$$

$$\mathbf{Z} = \hat{\mathbf{Z}}^{(L_2)}. \tag{D.10}$$

For a fair comparison, we employ a linear classifier as the base predictor for the C&S model in the experiment section's baseline model. This choice is motivated by the fact that the trainless version of the C&S model is also implemented with solely a linear classifier serving as the predictor.

D.2.3 Software and hardware details

We implement TrainlessGNN in Pytorch Geometric framework [50]. We run our experiments on a Linux system equipped with an NVIDIA A100 GPU.

D.2.4 Hyperparameter selections

Our method only has two hyperparameters governing the fitted weight matrix, which makes it easy to find the optimal hyperparameter setting. The two hyperparameters are the edge weight ω and the degree normalization matrix $\mathbf{R}$. Thus, we tune ω in $[-1, 0, 0.001, 0.01, 0.1, 1]$. For $\mathbf{R}$, we select the message passing types in Equation D.2 between CN, AA and RA. For the baseline methods, we train for 100 epochs and fine-tune the number of GNN layers, the learning rate, and the l-2 norm penalty. All the final result is selected based on the model's performance on the validation set.

D.2.5 Parameter Sensitivity

We further conduct a parameter sensitivity analysis to investigate the impact of the hyperparameters on TrainlessGNN. This analysis is performed using **Trainless SGC**, with the results shown in Figure D.2. The findings underscore that the optimal degree normalization matrix, $\mathbf{R}$, may vary across different benchmarks due to their unique data attributes. For example, RA-type message passing excels on the Pubmed and OGBN-Arxiv datasets, while CN and AA-type message passing proves more robust on the remaining datasets. Concurrently, the edge weight, ω, remains consistent across varied settings. However, a larger setting of ω ($\omega = 1$) can adversely affect the performance of our trainless methods by excessively penalizing the node attributes from other classes. Intriguingly, even with a slightly negative value of ω, our trainless model retains its efficacy. In such scenarios, the weight vector consists of not only the node attributes from the corresponding class but also from other classes. Yet, a continual reduction in ω leads to a marked performance decline, indicating that the weight vectors could become indistinguishable under such conditions.

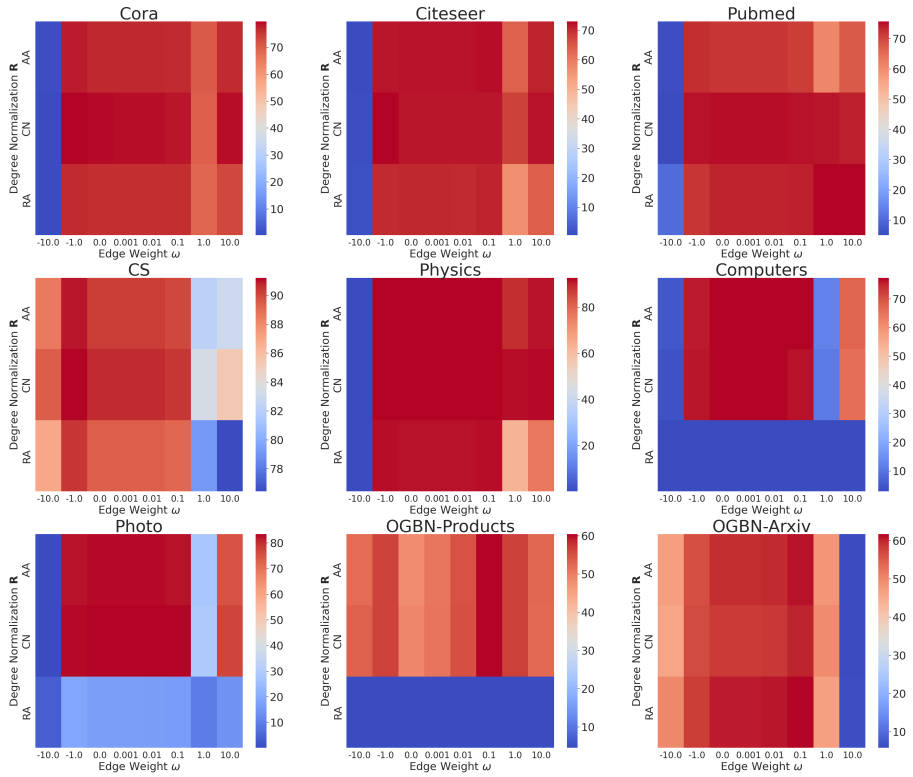

Figure D.2: Parameter sensitivity analysis for degree normalization matrix $\mathbf{R}$ and edge weight ω using the **Trainless SGC** model.

D.2.6 Experimental details on heterophilous graphs

On heterophilous graphs, we take the same experimental protocol as [23] to partition the datasets into 10 different splits. The splits are predefined as [29]. We run the same set of hyperparameter selections as the homophilous graph experiments.

We also conduct an experiment that fits the model including labels from validation sets. The results are shown in Figure D.3. We can see that the benefits of bringing more validation labels are marginal to TrainlessGNN on the heterophilous graphs.

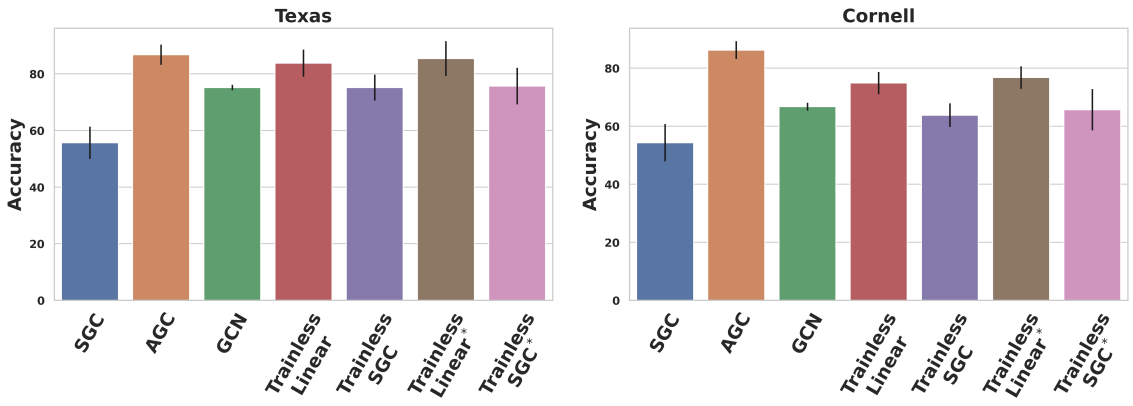

Figure D.3: Performance of our methods on heterophilous graphs. * indicates the models trained with training and validation labels.

D.3 Limitations

Despite the efficacy of our proposed method in tackling the semi-supervised node classification problem on text-attributed graphs, certain limitations prevail. Firstly, our method is confined to fitting a linear GNN model, which may limit the model's expressiveness. Secondly, the successful deployment of our method depends on specific data configurations, namely over-parameterization, potentially narrowing its applicability across a broader spectrum of cases.

APPENDIX E

UNIVERSAL LINK PREDICTOR BY IN-CONTEXT LEARNING ON GRAPHS

E.1 Experimental details

E.1.1 Pretrain and test benchmarks

Comprehensive details of the curated pretrain and test graph datasets are provided in Table E.1. These datasets, selected from various domains and featuring diverse graph statistics, are specifically chosen to ensure that our proposed UniLP model is exposed to a wide range of LP connectivity patterns. This diversity in training data is crucial for enabling UniLP to effectively adapt to new, unseen graphs.

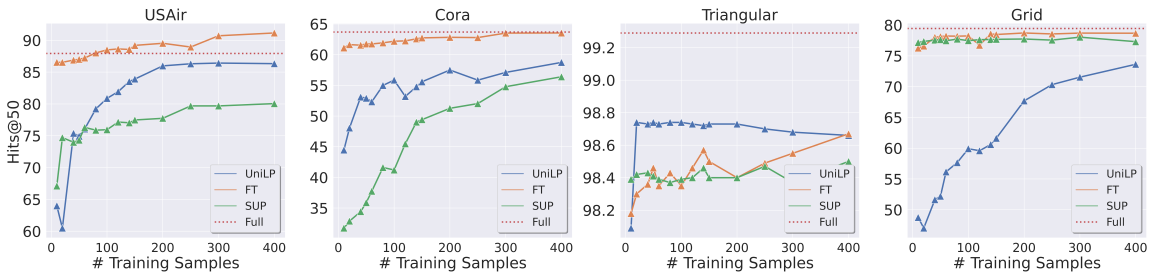

Figure E.1: Performance of UniLP with varying quantities of in-context links on the rest of graph datasets.

E.1.2 Pretraining the Models

We pretrain UniLP on the datasets listed in Table E.1, employing an approach by sampling an equal number of non-connected node pairs ($V \times V \backslash E^o$) as negative samples to match the count of observed links ($|E^o|$) in each graph. The pretraining follows a standard binary classification framework.

For predicting each query link, we sample 40 positive and negative links as in-context links ($S^+ \cup S^-$) from the respective pretrain dataset. This setup ensures variability: different query links from the same dataset or the same query link across training batches may be paired with different in-context links. However, during testing, the set of in-context links for each test dataset remains constant. This training methodology serves multiple purposes: it enhances UniLP's generalization capabilities by exposing it to a broad range of in-context links and optimizes GPU memory usage by selecting a manageable yet diverse set of in-context links during pretraining.

The pretraining phase incorporates an early stopping criterion based on performance across a merged validation set, which comprises 200 links from the validation set of each test dataset. This approach will stop UniLP's optimization once it reaches optimal performance on this merged validation set, ensuring efficiency and preventing overfitting.

E.1.3 Software and hardware details

We implement UniLP in Pytorch Geometric framework [50]. We conduct our experiments on a Linux system equipped with an NVIDIA A100 GPU with 80GB of memory.

E.2 Theoretical analysis

E.2.1 More discussions about the connectivity patterns

In the initial definition (Definition 5), connectivity patterns are characterized as ordered sequences of events that are satisfied by the features of links. This concept underpins the idea that if two graphs exhibit identical connectivity patterns, a LP model trained on one graph could theoretically be applied to the other without retraining. The rationale behind this is rooted in the LP task's core objective: to prioritize true links over false ones through ranking. Hence, a consistent ranking mechanism across different graphs allows for the same heuristic-based link predictor to be effectively utilized for LP tasks across those graphs.

It might be tempting to equate connectivity patterns directly with graph distributions; however, this is a misconception. Graphs can share identical connectivity patterns yet differ significantly in their underlying distributions. An illustrative example is provided by graphs generated through the Stochastic Block Model [60] with distinct parameters, which may still present identical connectivity patterns provided their intra-block edge probabilities are higher than those between blocks.

Consequently, despite real-world graphs often exhibiting varied underlying distributions—reflected in aspects such as node degrees, graph sizes, and densities—the question of whether a singular, common connectivity pattern exists across diverse graphs remains non-trivial. This inquiry forms the theoretical foundation for our Universal Link Predictor model, challenging us to explore the feasibility of applying one singular link prediction methodology in a world of inherently distinct graph structures.

E.2.2 Proof for Theorem 7

We first restate the theorem and proceed with the proof:

Define $A_2 = |\pi_2(u,v)| \geq 1$ and $A_3 = |\pi_3(u,v)| \geq 1$ as elements of ω. The connectivity patterns on Grid and Triangular graphs are distinct. Specifically:
(i) On Grid: $\omega = [A_3, A_2]$; (ii) On Triangular: $\omega = [A_2, A_3]$.

Proof. In a Grid graph, the probability of a connection given a 2-hop simple path, $p(y = 1|A_2)$, can be expressed as $\frac{p(y=1,A_2)}{p(A_2)}$. The absence of any 2-hop connected node pairs $(u,v) \in E^o$ implies $p(y = 1, A_2) = 0$, leading to $p(y = 1|A_2) = 0$.

Considering the symmetric nature of nodes in a synthetic Grid graph, we select an arbitrary node as an anchor. Identifying nodes with a 3-hop simple path to this anchor reveals that:

$$p(y = 1|A_3) = \frac{p(y = 1, A_3)}{p(A_3)} = \frac{4}{16} = \frac{1}{4}. \tag{E.1}$$

This calculation confirms the connectivity sequence on Grid as $\omega = [A_3, A_2]$.

Conversely, in a Triangular graph, the probabilities given a 2-hop and a 3-hop simple path are calculated as:

$$p(y = 1|A_2) = \frac{p(y = 1, A_2)}{p(A_2)} = \frac{6}{18} = \frac{1}{3},$$
$$p(y = 1|A_3) = \frac{p(y = 1, A_3)}{p(A_3)} = \frac{6}{36} = \frac{1}{6}.$$

Thus, establishing the connectivity sequence for Triangular as $\omega = [A_2, A_3]$, which is in direct contrast to that of Grid graphs, highlighting the inherent difference in their connectivity patterns. □

TABLE E.1

THE PRETRAIN DATASETS AND TEST BENCHMARKS.

Dataset	Pretrain	Test	# Nodes	# Edges	Avg. node deg.	Std. node deg.	Max. node deg.	Density
				Biology				
Ecoli	✓	-	1805	29320	16.24	48.38	1030	1.8009%
Yeast	✓	-	2375	23386	9.85	15.5	118	0.8295%
Celegans	-	✓	297	4296	14.46	12.97	134	9.7734%
				Transport				
Power	✓	-	4941	13188	2.67	1.79	19	0.1081%
USAir	-	✓	332	4252	12.81	20.13	139	7.7385%
				Web				
PolBlogs	✓	-	1490	19025	12.77	20.73	256	1.7150%
Router	✓	-	5022	12516	2.49	5.29	106	0.0993%
PB	-	✓	1222	33428	27.36	38.42	351	4.4808%
				Collaboration				
Physics	✓	-	34493	495924	14.38	15.57	382	0.0834%
CS	-	✓	18333	163788	8.93	9.11	136	0.0975%
NS	-	✓	1589	5484	3.45	3.47	34	0.4347%
				Citation				
Pubmed	✓	-	19717	88648	4.5	7.43	171	0.0456%
Citeseer	✓	-	3327	9104	2.74	3.38	99	0.1645%
Cora	-	✓	2708	10556	3.9	5.23	168	0.2880%
				Social				
Twitch	✓	-	34118	429113	12.58	35.88	1489	0.0737%
Github	✓	-	37700	289003	7.67	46.59	6809	0.0407%
Facebook	-	✓	22470	171002	7.61	15.26	472	0.0677%

APPENDIX F

ADAPTING GNNS TO RELATIONAL DATABASE

F.1 Technical details

F.1.1 `Txn-Bert` configuration

For the text encoding within `Rel-Cat`, we utilize the transformer architecture from Sentence-Bert [125], specifically leveraging the HuggingFace implementation of Sentence-Bert[1]. We primarily employ a 6-layer transformer configuration for `Rel-Cat` due to its efficiency and sufficiency in capturing relevant features from the transaction data. In our experiments, as detailed in Table 3.1, a 12-layer transformer was tested solely for ablation study purposes. The results indicated that the 6-layer model provided comparable performance with marginal benefits from the more complex 12-layer model.

F.1.2 Inference

In both the Zero Shot and Few Shot settings, we determine the scores by computing the inner product between the transaction and *Category* embeddings to identify the top 5 most likely predictions for a given transaction.

In Zero Shot setting, `Txn-Bert` encodes both the new transaction and all *Category* labels into text embeddings. The cosine similarities between the transaction embedding and each *Category* are then calculated, ranked, and the top 5 predictions are selected based on these rankings.

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

For the Few Shot setting, TopK NN is firstly employed to identify similar historical transactions for a given company, with a similarity threshold set at 0.8. The *Category* labels corresponding to these similar historical transactions are directly used as predictions. If fewer than 5 distinct *Category* are identified through this method, we engage the GNN component of `Rel-Cat` to compute embeddings for the transaction and *Category*. A ranking process is then conducted to select any remaining predictions needed to complete the top 5.

F.1.3 Software and Hardware details

We develop `Rel-Cat` using the PyTorch framework, PyTorch Geometric for graph neural network operations, and the HuggingFace library for transformer architectures. All experiments were conducted on an Amazon SageMaker instance equipped with four V100 GPUs.

F.2 Supplementary experiments

F.2.1 Prediction cascade

In the `Rel-Cat` pipeline shown in Figure 4.4, the processed graph initially enters the TopK NN module, which serves as an early exit strategy to identify similar transactions via text embeddings. If this step does not yield sufficient *Category* predictions, the pipeline advances to the GNNs for additional predictions.

Our analysis of this cascade, shown in Figure F.1, reveals that for Top 1 predictions, TopK NN alone efficiently handles over 68% of transactions. This high percentage underscores the prevalence of similar transactions within the database. However, to generate the top 5 predictions, more than 96% of transactions require processing by the GNNs, indicating the need for a more in-depth computation to fulfill broader prediction requirements.

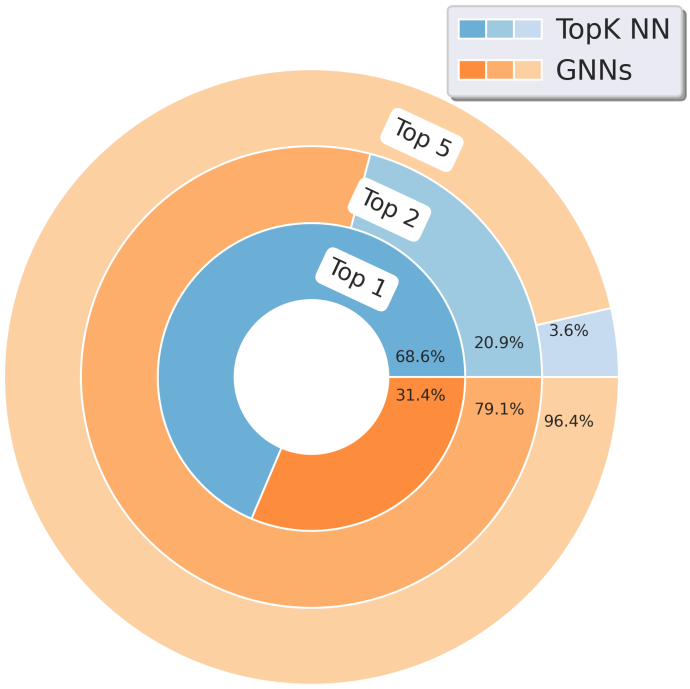

Figure F.1: Cascade Process in `Rel-Cat`. TopK NN efficiently resolves over 68% of transactions when only a Top 1 prediction is needed. However, for more comprehensive Top 5 predictions, over 96% of transactions necessitate processing by GNNs.

This flexible approach in `Rel-Cat` demonstrates the system's adaptability, allowing for a balance between efficiency and thoroughness in prediction based on system demands and user experience considerations.

BIBLIOGRAPHY

1. R. Abboud, I. I. Ceylan, M. Grohe, and T. Lukasiewicz. The Surprising Power of Graph Neural Networks with Random Node Initialization, 2021. _eprint: 2010.01179.

2. R. Abboud, R. Dimitrov, and I. I. Ceylan. Shortest Path Networks for Graph Property Prediction. Nov. 2022. URL https://openreview.net/forum?id=mWzWvMxuFg1.

3. R. Ackland and others. Mapping the US political blogosphere: Are conservative bloggers more prominent? In *BlogTalk Downunder 2005 Conference, Sydney*, 2005.

4. L. A. Adamic and E. Adar. Friends and neighbors on the Web. *Social Networks*, 25(3):211–230, 2003. ISSN 0378-8733. doi: https://doi.org/10.1016/S0378-8733(03)00009-1. URL https://www.sciencedirect.com/science/article/pii/S0378873303000091.

5. L. A. Adamic and N. Glance. The political blogosphere and the 2004 U.S. election: divided they blog. In *Proceedings of the 3rd international workshop on Link discovery*, LinkKDD '05, pages 36–43, New York, NY, USA, Aug. 2005. Association for Computing Machinery. ISBN 978-1-59593-215-0. doi: 10.1145/1134271.1134277. URL https://dl.acm.org/doi/10.1145/1134271.1134277.

6. A. A. Alemi, I. Fischer, J. V. Dillon, and K. Murphy. Deep Variational Information Bottleneck. Apr. 2023. URL https://openreview.net/forum?id=HyxQzBceg.

7. U. Alon and E. Yahav. On the Bottleneck of Graph Neural Networks and its Practical Implications. *arXiv:2006.05205 [cs, stat]*, Mar. 2021. URL http://arxiv.org/abs/2006.05205. arXiv: 2006.05205.

8. M. Arjovsky, L. Bottou, I. Gulrajani, and D. Lopez-Paz. Invariant Risk Minimization, July 2019. URL https://arxiv.org/abs/1907.02893v3.

9. A.-L. Barabási and R. Albert. Emergence of Scaling in Random Networks. *Science*, 286(5439):509–512, 1999. doi: 10.1126/science.286.5439.509. URL https://www.science.org/doi/abs/10.1126/science.286.5439.509. _eprint: https://www.science.org/doi/pdf/10.1126/science.286.5439.509.

10. V. Batagelj and A. Mrvar. Pajek datasets website, 2006. URL http://vlado.fmf.uni-lj.si/pub/networks/data/.

11. M. Belkin. Fit without fear: remarkable mathematical phenomena of deep learning through the prism of interpolation, May 2021. URL http://arxiv.org/abs/2105.14368. arXiv:2105.14368 [cs, math, stat].

12. J. Betker, G. Goh, L. Jing, T. Brooks, J. Wang, L. Li, L. Ouyang, J. Zhuang, J. Lee, Y. Guo, W. Manassra, P. Dhariwal, C. Chu, Y. Jiao, and A. Ramesh. Improving Image Generation with Better Captions. URL https://api.semanticscholar.org/CorpusID:264403242.

13. B. Bevilacqua, Y. Zhou, and B. Ribeiro. Size-Invariant Graph Representations for Graph Classification Extrapolations, 2021. URL https://arxiv.org/abs/2103.05045.

14. A. P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, July 1997. ISSN 0031-3203. doi: 10.1016/S0031-3203(96)00142-2. URL https://www.sciencedirect.com/science/article/pii/S0031320396001422.

15. S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks*, 30:107–117, 1998. URL http://www-db.stanford.edu/~backrub/google.html.

16. S. Brody, U. Alon, and E. Yahav. How Attentive are Graph Attention Networks?, Jan. 2022. URL http://arxiv.org/abs/2105.14491. arXiv:2105.14491 [cs].

17. M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, July 2017. ISSN 1053-5888, 1558-0792. doi: 10.1109/MSP.2017.2693418. URL http://arxiv.org/abs/1611.08097. arXiv:1611.08097.

18. T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language Models are Few-Shot Learners, July 2020. URL http://arxiv.org/abs/2005.14165. arXiv:2005.14165 [cs].

19. J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral Networks and Locally Connected Networks on Graphs. *arXiv:1312.6203 [cs]*, May 2014. URL http://arxiv.org/abs/1312.6203. arXiv: 1312.6203.

20. D. Buffelli, P. Liò, and F. Vandin. SizeShiftReg: a Regularization Method for Improving Size-Generalization in Graph Neural Networks, 2022. URL https://arxiv.org/abs/2207.07888.

21. Q. Cappart, D. Chételat, E. Khalil, A. Lodi, C. Morris, and P. Veličković. Combinatorial optimization and reasoning with graph neural networks, Sept. 2022. URL http://arxiv.org/abs/2102.09544. arXiv:2102.09544 [cs].

22. B. P. Chamberlain, S. Shirobokov, E. Rossi, F. Frasca, T. Markovich, N. Y. Hammerla, M. M. Bronstein, and M. Hansmire. Graph Neural Networks for Link Prediction with Subgraph Sketching. Sept. 2022. URL https://openreview.net/forum?id=m1oqEOAozQU.

23. S. Chanpuriya and C. N. Musco. Simplified Graph Convolution with Heterophily. May 2022. URL https://openreview.net/forum?id=jRrpiqxtrWm.

24. N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16:321–357, June 2002. ISSN 1076-9757. doi: 10.1613/jair.953. URL http://arxiv.org/abs/1106.1813. arXiv:1106.1813 [cs].

25. D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun. Measuring and Relieving the Over-smoothing Problem for Graph Neural Networks from the Topological View. In *AAAI*, 2020.

26. Y. Chen, L. Wu, and M. Zaki. Iterative Deep Graph Learning for Graph Neural Networks: Better and Robust Node Embeddings. In *Advances in Neural Information Processing Systems*, volume 33, pages 19314–19326. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/hash/e05c7ba4e087beea9410929698dc41a6-Abstract.html.

27. Z. Chen, L. Chen, S. Villar, and J. Bruna. Can Graph Neural Networks Count Substructures? *arXiv:2002.04025 [cs, stat]*, Oct. 2020. URL http://arxiv.org/abs/2002.04025. arXiv: 2002.04025.

28. Z. Chen, H. Mao, H. Li, W. Jin, H. Wen, X. Wei, S. Wang, D. Yin, W. Fan, H. Liu, and J. Tang. Exploring the Potential of Large Language Models (LLMs) in Learning on Graphs, Aug. 2023. URL http://arxiv.org/abs/2307.03393. arXiv:2307.03393 [cs].

29. M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. Learning to construct knowledge bases from the World Wide Web. *Artificial Intelligence*, 118(1):69–113, Apr. 2000. ISSN 0004-3702. doi: 10.1016/S0004-3702(00)00004-7. URL https://www.sciencedirect.com/science/article/pii/S0004370200000047.

30. E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le. AutoAugment: Learning Augmentation Policies from Data, Apr. 2019. URL http://arxiv.org/abs/1805.09501. arXiv:1805.09501 [cs, stat].

31. D. Dai, Y. Sun, L. Dong, Y. Hao, S. Ma, Z. Sui, and F. Wei. Why Can GPT Learn In-Context? Language Models Implicitly Perform Gradient Descent as Meta-Optimizers, Dec. 2022. URL https://arxiv.org/abs/2212.10559v3.

32. DeepSeek-AI, D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, X. Zhang, X. Yu, Y. Wu, Z. F. Wu, Z. Gou, Z. Shao, Z. Li, Z. Gao, A. Liu, B. Xue, B. Wang, B. Wu, B. Feng, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, D. Dai, D. Chen, D. Ji, E. Li, F. Lin, F. Dai, F. Luo, G. Hao, G. Chen, G. Li, H. Zhang, H. Bao, H. Xu, H. Wang, H. Ding, H. Xin, H. Gao, H. Qu, H. Li, J. Guo, J. Li, J. Wang, J. Chen, J. Yuan, J. Qiu, J. Li, J. L. Cai, J. Ni, J. Liang, J. Chen, K. Dong, K. Hu, K. Gao, K. Guan, K. Huang, K. Yu, L. Wang, L. Zhang, L. Zhao, L. Wang, L. Zhang, L. Xu, L. Xia, M. Zhang, M. Zhang, M. Tang, M. Li, M. Wang, M. Li, N. Tian, P. Huang, P. Zhang, Q. Wang, Q. Chen, Q. Du, R. Ge, R. Zhang, R. Pan, R. Wang, R. J. Chen, R. L. Jin, R. Chen, S. Lu, S. Zhou, S. Chen, S. Ye, S. Wang, S. Yu, S. Zhou, S. Pan, S. S. Li, S. Zhou, S. Wu, S. Ye, T. Yun, T. Pei, T. Sun, T. Wang, W. Zeng, W. Zhao, W. Liu, W. Liang, W. Gao, W. Yu, W. Zhang, W. L. Xiao, W. An, X. Liu, X. Wang, X. Chen, X. Nie, X. Cheng, X. Liu, X. Xie, X. Liu, X. Yang, X. Li, X. Su, X. Lin, X. Q. Li, X. Jin, X. Shen, X. Chen, X. Sun, X. Wang, X. Song, X. Zhou, X. Wang, X. Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. Zhang, Y. Xu, Y. Li, Y. Zhao, Y. Sun, Y. Wang, Y. Yu, Y. Zhang, Y. Shi, Y. Xiong, Y. He, Y. Piao, Y. Wang, Y. Tan, Y. Ma, Y. Liu, Y. Guo, Y. Ou, Y. Wang, Y. Gong, Y. Zou, Y. He, Y. Xiong, Y. Luo, Y. You, Y. Liu, Y. Zhou, Y. X. Zhu, Y. Xu, Y. Huang, Y. Li, Y. Zheng, Y. Zhu, Y. Ma, Y. Tang, Y. Zha, Y. Yan, Z. Z. Ren, Z. Ren, Z. Sha, Z. Fu, Z. Xu, Z. Xie, Z. Zhang, Z. Hao, Z. Ma, Z. Yan, Z. Wu, Z. Gu, Z. Zhu, Z. Liu, Z. Li, Z. Xie, Z. Song, Z. Pan, Z. Huang, Z. Xu, Z. Zhang, and Z. Zhang. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning, Jan. 2025. URL http://arxiv.org/abs/2501.12948. arXiv:2501.12948 [cs].

33. M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper/2016/file/04df4d434d481c5bb723be1b6df1ee65-Paper.pdf.

34. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In J. Burstein, C. Doran, and T. Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

35. T. DeVries and G. W. Taylor. Improved Regularization of Convolutional Neural

Networks with Cutout, Nov. 2017. URL http://arxiv.org/abs/1708.04552. arXiv:1708.04552 [cs].

36. J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition. In E. P. Xing and T. Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 647–655, Bejing, China, June 2014. PMLR. URL https://proceedings.mlr.press/v32/donahue14.html. Issue: 1.

37. K. Dong, Y. Tian, Z. Guo, Y. Yang, and N. Chawla. FakeEdge: Alleviate Dataset Shift in Link Prediction. Dec. 2022. URL https://openreview.net/forum?id=QDN0jSXuvtX.

38. K. Dong, Z. Guo, and N. V. Chawla. Pure Message Passing Can Estimate Common Neighbor for Link Prediction, Oct. 2023. URL http://arxiv.org/abs/2309.00976. arXiv:2309.00976 [cs].

39. K. Dong, Z. Guo, and N. V. Chawla. CORE: Data Augmentation for Link Prediction via Information Bottleneck, Apr. 2024. URL http://arxiv.org/abs/2404.11032. arXiv:2404.11032 [cs].

40. K. Dong, Z. Guo, and N. V. Chawla. You do not have to train Graph Neural Networks at all on text-attributed graphs, Apr. 2024. URL http://arxiv.org/abs/2404.11019. arXiv:2404.11019 [cs].

41. K. Dong, H. Mao, Z. Guo, and N. V. Chawla. Universal Link Predictor By In-Context Learning on Graphs, Feb. 2024. URL http://arxiv.org/abs/2402.07738. arXiv:2402.07738 [cs].

42. M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou. The Faiss library. 2024. _eprint: 2401.08281.

43. D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional Networks on Graphs for Learning Molecular Fingerprints. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper/2015/file/f9be311e65d81a9ad8150a60844bb94c-Paper.pdf.

44. V. P. Dwivedi, C. K. Joshi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson. Benchmarking Graph Neural Networks, Dec. 2022. URL http://arxiv.org/abs/2003.00982. arXiv:2003.00982 [cs].

45. D. Easley, J. Kleinberg, and others. *Networks, crowds, and markets: Reasoning about a highly connected world*, volume 1. Cambridge university press Cambridge, 2010.

46. P. Erdos. On random graphs. *Publicationes mathematicae*, 6:290–297, 1959.

47. J. Feng, Y. Chen, F. Li, A. Sarkar, and M. Zhang. How Powerful are K-hop Message Passing Graph Neural Networks. May 2022. URL https://openreview.net/forum?id=nN3aVRQsxGd.

48. J. Feng, Y. Chen, F. Li, A. Sarkar, and M. Zhang. How Powerful are K-hop Message Passing Graph Neural Networks, Jan. 2023. URL http://arxiv.org/abs/2205.13328. arXiv:2205.13328 [cs].

49. S. Y. Feng, V. Gangal, J. Wei, S. Chandar, S. Vosoughi, T. Mitamura, and E. Hovy. A Survey of Data Augmentation Approaches for NLP. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 968–988, Online, Aug. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-acl.84. URL https://aclanthology.org/2021.findings-acl.84.

50. M. Fey and J. E. Lenssen. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

51. M. Fey, W. Hu, K. Huang, J. E. Lenssen, R. Ranjan, J. Robinson, R. Ying, J. You, and J. Leskovec. Position: Relational Deep Learning - Graph Representation Learning on Relational Databases. June 2024. URL https://openreview.net/forum?id=BIMSHniyCP&referrer=%5Bthe%20profile%20of%20Jan%20Eric%20Lenssen%5D(%2Fprofile%3Fid%3D~Jan_Eric_Lenssen1).

52. J. Frankle and M. Carbin. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. Sept. 2018. URL https://openreview.net/forum?id=rJl-b3RcF7.

53. F. Frasca, E. Rossi, D. Eynard, B. Chamberlain, M. Bronstein, and F. Monti. SIGN: Scalable Inception Graph Neural Networks, Nov. 2020. URL http://arxiv.org/abs/2004.11198. arXiv:2004.11198 [cs, stat].

54. F. Frasca, B. Bevilacqua, M. M. Bronstein, and H. Maron. Understanding and Extending Subgraph GNNs by Rethinking Their Symmetries, June 2022. URL http://arxiv.org/abs/2206.11140. arXiv:2206.11140 [cs].

55. A. Ghasemian, H. Hosseinmardi, A. Galstyan, E. M. Airoldi, and A. Clauset. Stacking models for nearly optimal link prediction in complex networks. *Proceedings of the National Academy of Sciences*, 117(38):23393–23400, 2020. doi: 10.1073/pnas.1914950117. URL https://www.pnas.org/doi/abs/10.1073/pnas.1914950117. _eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.1914950117.

56. C. L. Giles, K. D. Bollacker, and S. Lawrence. CiteSeer: An automatic citation indexing system. In *Proceedings of the third ACM conference on Digital libraries*, pages 89–98, 1998.

57. J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural Message Passing for Quantum Chemistry. *CoRR*, abs/1704.01212, 2017. URL http://arxiv.org/abs/1704.01212. arXiv: 1704.01212.

58. Z. Guo, W. Shiao, S. Zhang, Y. Liu, N. Chawla, N. Shah, and T. Zhao. Linkless Link Prediction via Relational Distillation. *arXiv preprint arXiv:2210.05801*, 2022.

59. W. L. Hamilton, R. Ying, and J. Leskovec. Inductive Representation Learning on Large Graphs. *arXiv:1706.02216 [cs, stat]*, Sept. 2018. URL http://arxiv.org/abs/1706.02216. arXiv: 1706.02216.

60. P. Holland, K. B. Laskey, and S. Leinhardt. Stochastic blockmodels: First steps. *Social Networks*, 5:109–137, 1983. URL https://api.semanticscholar.org/CorpusID:34098453.

61. K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, Jan. 1989. ISSN 0893-6080. doi: 10.1016/0893-6080(89)90020-8. URL https://www.sciencedirect.com/science/article/pii/0893608089900208.

62. W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *arXiv:2005.00687 [cs, stat]*, Feb. 2021. URL http://arxiv.org/abs/2005.00687. arXiv: 2005.00687.

63. Y. Hu, X. Wang, Z. Lin, P. Li, and M. Zhang. Two-Dimensional Weisfeiler-Lehman Graph Neural Networks for Link Prediction, June 2022. URL http://arxiv.org/abs/2206.09567. arXiv:2206.09567 [cs].

64. Q. Huang, H. He, A. Singh, S.-N. Lim, and A. Benson. Combining Label Propagation and Simple Models out-performs Graph Neural Networks. Oct. 2020. URL https://openreview.net/forum?id=8E1-f3VhX1o.

65. Q. Huang, H. Ren, P. Chen, G. Kržmanc, D. Zeng, P. Liang, and J. Leskovec. PRODIGY: Enabling In-context Learning Over Graphs. Nov. 2023. URL https://openreview.net/forum?id=pLwYhNNnoR.

66. T. Huang, T. Chen, M. Fang, V. Menkovski, J. Zhao, L. Yin, Y. Pei, D. C. Mocanu, Z. Wang, M. Pechenizkiy, and S. Liu. You Can Have Better Graph Neural Networks by Not Training Weights at All: Finding Untrained GNNs Tickets. Nov. 2022. URL https://openreview.net/forum?id=dF6aEW3_62O.

67. E. Hwang, V. Thost, S. S. Dasgupta, and T. Ma. An Analysis of Virtual Nodes in Graph Neural Networks for Link Prediction (Extended Abstract). Nov. 2022. URL https://openreview.net/forum?id=dI6KBKNRp7.

68. K. Irie, R. Csordás, and J. Schmidhuber. The Dual Form of Neural Networks Revisited: Connecting Test Time Predictions to Training Patterns via Spotlights of Attention, June 2022. URL http://arxiv.org/abs/2202.05798. arXiv:2202.05798 [cs].

69. P. Jaccard. The Distribution of the Flora in the Alpine Zone.1. *New Phytologist*, 11(2):37–50, 1912. ISSN 1469-8137. doi: 10.1111/j.1469-8137.1912.tb05611.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1469-8137.1912.tb05611.x. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1469-8137.1912.tb05611.x.

70. E. Jang, S. Gu, and B. Poole. Categorical Reparameterization with Gumbel-Softmax. Apr. 2023. URL https://openreview.net/forum?id=rkE3y85ee.

71. A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed. Mistral 7B, Oct. 2023. URL http://arxiv.org/abs/2310.06825. arXiv:2310.06825 [cs].

72. J. Jin, Y. Wang, W. Zhang, Q. Gan, X. Song, Y. Yu, Z. Zhang, and D. Wipf. Refined Edge Usage of Graph Neural Networks for Edge Prediction. Dec. 2022. doi: 10.48550/arXiv.2212.12970. URL https://arxiv.org/abs/2212.12970v1.

73. W. Johnson and J. Lindenstrauss. Extensions of Lipschitz maps into a Hilbert space. *Contemporary Mathematics*, 26:189–206, Jan. 1984. ISSN 9780821850305. doi: 10.1090/conm/026/737400.

74. P. C. Kainen. Orthogonal dimension and tolerance. *Unpublished report, Washington DC: Industrial Math*, 1992.

75. P. C. Kainen and V. Kurkova. Quasiorthogonal dimension. In *Beyond traditional probabilistic data processing techniques: Interval, fuzzy etc. Methods and their applications*, pages 615–629. Springer, 2020.

76. P. C. Kainen and V. Kůrková. Quasiorthogonal dimension of euclidean spaces. *Applied Mathematics Letters*, 6(3):7–10, May 1993. ISSN 0893-9659. doi: 10.1016/0893-9659(93)90023-G. URL https://www.sciencedirect.com/science/article/pii/089396599390023G.

77. J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling Laws for Neural Language Models, Jan. 2020. URL http://arxiv.org/abs/2001.08361. arXiv:2001.08361 [cs].

78. L. Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, Mar. 1953. ISSN 1860-0980. doi: 10.1007/BF02289026. URL https://doi.org/10.1007/BF02289026.

79. D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*, May 2014. URL http://arxiv.org/abs/1312.6114. arXiv: 1312.6114.

80. D. P. Kingma, D. J. Rezende, S. Mohamed, and M. Welling. Semi-Supervised Learning with Deep Generative Models, Oct. 2014. URL http://arxiv.org/abs/1406.5298. arXiv:1406.5298 [cs, stat].

81. T. N. Kipf and M. Welling. Variational Graph Auto-Encoders, 2016. _eprint: 1611.07308.

82. T. N. Kipf and M. Welling. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv:1609.02907 [cs, stat]*, Feb. 2017. URL http://arxiv.org/abs/1609.02907. arXiv: 1609.02907.

83. A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick. Segment Anything, Apr. 2023. URL http://arxiv.org/abs/2304.02643. arXiv:2304.02643 [cs].

84. J. Klicpera, S. Weißenberger, and S. Günnemann. Diffusion Improves Graph Learning. *arXiv:1911.05485 [cs, stat]*, Dec. 2019. URL http://arxiv.org/abs/1911.05485. arXiv: 1911.05485.

85. Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009. Publisher: IEEE.

86. D. Krueger, E. Caballero, J.-H. Jacobsen, A. Zhang, J. Binas, D. Zhang, R. L. Priol, and A. Courville. Out-of-Distribution Generalization via Risk Extrapolation (REx), Mar. 2020. URL https://arxiv.org/abs/2003.00688v5.

87. A. Kumar, S. S. Singh, K. Singh, and B. Biswas. Link prediction techniques, applications, and performance: A survey. *Physica A: Statistical Mechanics and its Applications*, 553:124289, Sept. 2020. ISSN 0378-4371. doi: 10.1016/j.physa.2020.124289. URL https://www.sciencedirect.com/science/article/pii/S0378437120300856.

88. C. Lesner, A. Ran, M. Rukonic, and W. Wang. Large Scale Personalized Categorization of Financial Transactions. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):9365–9372, July 2019. ISSN 2374-3468. doi: 10.1609/aaai.v33i01.33019365. URL https://ojs.aaai.org/index.php/AAAI/article/view/4984. Number: 01.

89. L. Li and M. Spratling. Data Augmentation Alone Can Improve Adversarial Training, Jan. 2023. URL https://arxiv.org/abs/2301.09879v1.

90. P. Li, Y. Wang, H. Wang, and J. Leskovec. Distance Encoding: Design Provably More Powerful Neural Networks for Graph Representation Learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 4465–4478. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/2f73168bf3656f697507752ec592c437-Paper.pdf.

91. D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. In *Proceedings of the twelfth international conference on Information and knowledge management*, CIKM '03, pages 556–559, New York, NY, USA, Nov. 2003. Association for Computing Machinery. ISBN 978-1-58113-723-1. doi: 10.1145/956863.956972. URL http://doi.org/10.1145/956863.956972.

92. J. Liu, L. Pei, Y. Sun, H. Simpson, J. Lu, and N. Ho. Categorization of Financial Transactions in QuickBooks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, pages 3299–3307, New York, NY, USA, Aug. 2021. Association for Computing Machinery. ISBN 978-1-4503-8332-5. doi: 10.1145/3447548.3467100. URL https://doi.org/10.1145/3447548.3467100.

93. L. Lu and T. Zhou. Link Prediction in Complex Networks: A Survey. *Physica A: Statistical Mechanics and its Applications*, 390(6):1150–1170, Mar. 2011. ISSN 03784371. doi: 10.1016/j.physa.2010.11.027. URL http://arxiv.org/abs/1010.0725. arXiv:1010.0725 [physics].

94. Y. Luo, M. McThrow, W. Y. Au, T. Komikado, K. Uchino, K. Maruhashi, and S. Ji. Automated Data Augmentations for Graph Classification, Feb. 2023. URL http://arxiv.org/abs/2202.13248. arXiv:2202.13248 [cs].

95. M.-T. Luong, H. Pham, and C. D. Manning. Effective Approaches to Attention-based Neural Machine Translation, 2015. URL https://arxiv.org/abs/1508.04025.

96. C. J. Maddison, A. Mnih, and Y. W. Teh. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. Apr. 2023. URL https://openreview.net/forum?id=S1jE5L5gl.

97. H. Mao, J. Li, H. Shomer, B. Li, W. Fan, Y. Ma, T. Zhao, N. Shah, and J. Tang. Revisiting Link Prediction: A Data Perspective, Oct. 2023. URL http://arxiv.org/abs/2310.00793. arXiv:2310.00793 [cs].

98. H. Mao, Z. Chen, W. Tang, J. Zhao, Y. Ma, T. Zhao, N. Shah, M. Galkin, and J. Tang. Position: graph foundation models are already here. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *ICML'24*, pages 34670–34692, Vienna, Austria, July 2024. JMLR.org.

99. H. Mao, G. Liu, Y. Ma, R. Wang, and J. Tang. A Data Generation Perspective to the Mechanism of In-Context Learning, Feb. 2024. URL http://arxiv.org/abs/2402.02212. arXiv:2402.02212 [cs].

100. H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman. Provably Powerful Graph Networks. *arXiv:1905.11136 [cs, stat]*, June 2020. URL http://arxiv.org/abs/1905.11136. arXiv: 1905.11136.

101. S. Maskey, R. Levie, Y. Lee, and G. Kutyniok. Generalization Analysis of Message Passing Neural Networks on Large Random Graphs, 2022. URL https://arxiv.org/abs/2202.00645.

102. A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000. Publisher: Springer.

103. S. Miao, M. Liu, and P. Li. Interpretable and Generalizable Graph Learning via Stochastic Attention Mechanism, June 2022. URL http://arxiv.org/abs/2201.12987. arXiv:2201.12987 [cs].

104. T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed Representations of Words and Phrases and their Compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.

105. R. Milo, S. Itzkovitz, N. Kashtan, R. Levitt, S. Shen-Orr, I. Ayzenshtat, M. Sheffer, and U. Alon. Superfamilies of Evolved and Designed Networks. *Science*, 303(5663):1538–1542, 2004. doi: 10.1126/science.1089167. URL https://www.science.org/doi/abs/10.1126/science.1089167. _eprint: https://www.science.org/doi/pdf/10.1126/science.1089167.

106. S. Min, X. Lyu, A. Holtzman, M. Artetxe, M. Lewis, H. Hajishirzi, and L. Zettlemoyer. Rethinking the Role of Demonstrations: What Makes In-Context Learning Work?, Oct. 2022. URL http://arxiv.org/abs/2202.12837. arXiv:2202.12837 [cs].

107. J. G. Moreno-Torres, T. Raeder, R. Alaiz-RodríGuez, N. V. Chawla, and F. Herrera. A unifying view on dataset shift in classification. *Pattern Recognition*, 45 (1):521–530, Jan. 2012. ISSN 0031-3203. doi: 10.1016/j.patcog.2011.06.019. URL http://doi.org/10.1016/j.patcog.2011.06.019.

108. C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks, Nov. 2021. URL http://arxiv.org/abs/1810.02244. arXiv:1810.02244 [cs, stat].

109. G. Namata, B. London, L. Getoor, and B. Huang. Query-driven active surveying for collective classification. In *10th International Workshop on Mining and Learning with Graphs*, volume 8, page 1, 2012.

110. M. E. Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical review E*, 74(3):036104, 2006. Publisher: APS.

111. M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103 (23):8577–8582, 2006. doi: 10.1073/pnas.0601602103. URL https://www.pnas.org/doi/abs/10.1073/pnas.0601602103. _eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.0601602103.

112. Z. Niu, G. Zhong, and H. Yu. A review on the attention mechanism of deep learning. *Neurocomputing*, 452:48–62, Sept. 2021. ISSN 0925-2312. doi: 10.1016/j.neucom.2021.03.091. URL https://www.sciencedirect.com/science/article/pii/S092523122100477X.

113. H. NT and T. Maehara. Revisiting Graph Neural Networks: All We Have is Low-Pass Filters, May 2019. URL http://arxiv.org/abs/1905.09550. arXiv:1905.09550 [cs, math, stat].

114. I. Nunes, M. Heddes, P. Vergés, D. Abraham, A. Veidenbaum, A. Nicolau, and T. Givargis. DotHash: Estimating Set Similarity Metrics for Link Prediction and Document Deduplication, May 2023. URL http://arxiv.org/abs/2305.17310. arXiv:2305.17310 [cs].

115. OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, R. Avila, I. Babuschkin, S. Balaji, V. Balcom, P. Baltescu, H. Bao, M. Bavarian, J. Belgum, I. Bello, J. Berdine, G. Bernadett-Shapiro, C. Berner, L. Bogdonoff, O. Boiko, M. Boyd, A.-L. Brakman, G. Brockman, T. Brooks, M. Brundage, K. Button, T. Cai, R. Campbell, A. Cann, B. Carey, C. Carlson, R. Carmichael, B. Chan, C. Chang, F. Chantzis, D. Chen, S. Chen, R. Chen, J. Chen, M. Chen, B. Chess, C. Cho, C. Chu, H. W. Chung, D. Cummings, J. Currier, Y. Dai, C. Decareaux, T. Degry, N. Deutsch, D. Deville, A. Dhar, D. Dohan, S. Dowling, S. Dunning, A. Ecoffet, A. Eleti, T. Eloundou, D. Farhi, L. Fedus, N. Felix, S. P. Fishman, J. Forte, I. Fulford, L. Gao, E. Georges, C. Gibson, V. Goel, T. Gogineni, G. Goh, R. Gontijo-Lopes, J. Gordon, M. Grafstein, S. Gray, R. Greene, J. Gross, S. S. Gu, Y. Guo, C. Hallacy, J. Han, J. Harris, Y. He, M. Heaton, J. Heidecke, C. Hesse, A. Hickey, W. Hickey, P. Hoeschele, B. Houghton, K. Hsu, S. Hu, X. Hu, J. Huizinga, S. Jain, S. Jain, J. Jang, A. Jiang, R. Jiang, H. Jin, D. Jin, S. Jomoto, B. Jonn, H. Jun, T. Kaftan, L. Kaiser, A. Kamali, I. Kanitscheider, N. S. Keskar, T. Khan, L. Kilpatrick, J. W. Kim, C. Kim, Y. Kim, J. H. Kirchner, J. Kiros, M. Knight, D. Kokotajlo, L. Kondraciuk, A. Kondrich, A. Konstantinidis, K. Kosic, G. Krueger, V. Kuo, M. Lampe, I. Lan, T. Lee, J. Leike, J. Leung, D. Levy, C. M. Li, R. Lim, M. Lin, S. Lin, M. Litwin,

T. Lopez, R. Lowe, P. Lue, A. Makanju, K. Malfacini, S. Manning, T. Markov, Y. Markovski, B. Martin, K. Mayer, A. Mayne, B. McGrew, S. M. McKinney, C. McLeavey, P. McMillan, J. McNeil, D. Medina, A. Mehta, J. Menick, L. Metz, A. Mishchenko, P. Mishkin, V. Monaco, E. Morikawa, D. Mossing, T. Mu, M. Murati, O. Murk, D. Mély, A. Nair, R. Nakano, R. Nayak, A. Neelakantan, R. Ngo, H. Noh, L. Ouyang, C. O'Keefe, J. Pachocki, A. Paino, J. Palermo, A. Pantuliano, G. Parascandolo, J. Parish, E. Parparita, A. Passos, M. Pavlov, A. Peng, A. Perelman, F. d. A. B. Peres, M. Petrov, H. P. d. O. Pinto, Michael, Pokorny, M. Pokrass, V. H. Pong, T. Powell, A. Power, B. Power, E. Proehl, R. Puri, A. Radford, J. Rae, A. Ramesh, C. Raymond, F. Real, K. Rimbach, C. Ross, B. Rotsted, H. Roussez, N. Ryder, M. Saltarelli, T. Sanders, S. Santurkar, G. Sastry, H. Schmidt, D. Schnurr, J. Schulman, D. Selsam, K. Sheppard, T. Sherbakov, J. Shieh, S. Shoker, P. Shyam, S. Sidor, E. Sigler, M. Simens, J. Sitkin, K. Slama, I. Sohl, B. Sokolowsky, Y. Song, N. Staudacher, F. P. Such, N. Summers, I. Sutskever, J. Tang, N. Tezak, M. B. Thompson, P. Tillet, A. Tootoonchian, E. Tseng, P. Tuggle, N. Turley, J. Tworek, J. F. C. Uribe, A. Vallone, A. Vijayvergiya, C. Voss, C. Wainwright, J. J. Wang, A. Wang, B. Wang, J. Ward, J. Wei, C. J. Weinmann, A. Welihinda, P. Welinder, J. Weng, L. Weng, M. Wiethoff, D. Willner, C. Winter, S. Wolrich, H. Wong, L. Workman, S. Wu, J. Wu, M. Wu, K. Xiao, T. Xu, S. Yoo, K. Yu, Q. Yuan, W. Zaremba, R. Zellers, C. Zhang, M. Zhang, S. Zhao, T. Zheng, J. Zhuang, W. Zhuk, and B. Zoph. GPT-4 Technical Report, Mar. 2024. URL `http://arxiv.org/abs/2303.08774`. arXiv:2303.08774 [cs].

116. L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking : Bringing Order to the Web. In *The Web Conference*, 1999. URL `https://api.semanticscholar.org/CorpusID:1508503`.

117. J. Pan, T. Gao, H. Chen, and D. Chen. What In-Context Learning "Learns" In-Context: Disentangling Task Recognition and Task Learning, May 2023. URL `http://arxiv.org/abs/2305.09731`. arXiv:2305.09731 [cs].

118. L. Pan, C. Shi, and I. Dokmanić. Neural Link Prediction with Walk Pooling. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=CCu6RcUMwK0`.

119. P. Panda, A. Sengupta, and K. Roy. Conditional deep learning for energy-efficient and enhanced pattern recognition. In *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, DATE '16, pages 475–480, San Jose, CA, USA, Mar. 2016. EDA Consortium. ISBN 978-3-9815370-6-2.

120. P. A. Papp, K. Martinkus, L. Faber, and R. Wattenhofer. DropGNN: Random Dropouts Increase the Expressiveness of Graph Neural Networks, Nov. 2021. URL `http://arxiv.org/abs/2111.06283`. arXiv:2111.06283 [cs].

121. P. M. Phothilimthana, S. Abu-El-Haija, K. Cao, B. Fatemi, M. Burrows, C. Mendis, and B. Perozzi. TpuGraphs: A Performance Prediction Dataset on

Large Tensor Computational Graphs. Nov. 2023. URL https://openreview.net/forum?id=plAix1NxhU.

122. S. Purchase, Y. Zhao, and R. D. Mullins. Revisiting Embeddings for Graph Neural Networks. Nov. 2022. URL https://openreview.net/forum?id=Ri2dzVt_a1h.

123. J. Quiñonero-Candela, M. Sugiyama, A. Schwaighofer, N. D. Lawrence, M. I. Jordan, and T. G. Dietterich, editors. *Dataset Shift in Machine Learning*. Neural Information Processing series. MIT Press, Cambridge, MA, USA, Dec. 2008. ISBN 978-0-262-17005-5.

124. A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning Transferable Visual Models From Natural Language Supervision. In *Proceedings of the 38th International Conference on Machine Learning*, pages 8748–8763. PMLR, July 2021. URL https://proceedings.mlr.press/v139/radford21a.html. ISSN: 2640-3498.

125. N. Reimers and I. Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In K. Inui, J. Jiang, V. Ng, and X. Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China, Nov. 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1410. URL https://aclanthology.org/D19-1410.

126. J. Robinson, R. Ranjan, W. Hu, K. Huang, J. Han, A. Dobles, M. Fey, J. E. Lenssen, Y. Yuan, Z. Zhang, X. He, and J. Leskovec. RelBench: A Benchmark for Deep Learning on Relational Databases, July 2024. URL http://arxiv.org/abs/2407.20060. arXiv:2407.20060 [cs].

127. Y. Rong, W. Huang, T. Xu, and J. Huang. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=Hkx1qkrKPr.

128. B. Rozemberczki, C. Allen, and R. Sarkar. Multi-Scale attributed node embedding. *Journal of Complex Networks*, 9(2):cnab014, May 2021. ISSN 2051-1329. doi: 10.1093/comnet/cnab014. URL https://doi.org/10.1093/comnet/cnab014. _eprint: https://academic.oup.com/comnet/article-pdf/9/2/cnab014/40435146/cnab014.pdf.

129. R. Sato, M. Yamada, and H. Kashima. Random Features Strengthen Graph Neural Networks, 2021. _eprint: 2002.03155.

130. M. Schuster and K. Nakajima. Japanese and Korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*,

pages 5149–5152, Mar. 2012. doi: 10.1109/ICASSP.2012.6289079. URL https://ieeexplore-ieee-org.proxy.library.nd.edu/document/6289079. ISSN: 2379-190X.

131. O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann. Pitfalls of Graph Neural Network Evaluation, June 2019. URL http://arxiv.org/abs/1811.05868. arXiv:1811.05868 [cs, stat].

132. N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

133. C. Shorten and T. M. Khoshgoftaar. A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, 6(1):60, July 2019. ISSN 2196-1115. doi: 10.1186/s40537-019-0197-0. URL https://doi.org/10.1186/s40537-019-0197-0.

134. A. Singh, Q. Huang, S. L. Huang, O. Bhalerao, H. He, S.-N. Lim, and A. R. Benson. Edge Proposal Sets for Link Prediction, June 2021. URL http://arxiv.org/abs/2106.15810. arXiv:2106.15810 [cs].

135. K. K. Singh, H. Yu, A. Sarmasi, G. Pradeep, and Y. J. Lee. Hide-and-Seek: A Data Augmentation Technique for Weakly-Supervised Localization and Beyond, Nov. 2018. URL http://arxiv.org/abs/1811.02545. arXiv:1811.02545 [cs].

136. N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with Rocketfuel. *ACM SIGCOMM Computer Communication Review*, 32(4):133–145, 2002. Publisher: ACM New York, NY, USA.

137. B. Srinivasan and B. Ribeiro. On the Equivalence between Positional Node Embeddings and Structural Graph Representations. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SJxzFySKwH.

138. Q. Sun, J. Li, H. Peng, J. Wu, X. Fu, C. Ji, and P. S. Yu. Graph Structure Learning with Variational Information Bottleneck, Dec. 2021. URL http://arxiv.org/abs/2112.08903. arXiv:2112.08903 [cs].

139. S. Sun, Z. Cao, H. Zhu, and J. Zhao. A Survey of Optimization Methods from a Machine Learning Perspective, Oct. 2019. URL http://arxiv.org/abs/1906.06821. arXiv:1906.06821 [cs, math, stat].

140. S. Suresh, P. Li, C. Hao, and J. Neville. Adversarial Graph Augmentation to Improve Graph Contrastive Learning, Nov. 2021. URL http://arxiv.org/abs/2106.05819. arXiv:2106.05819 [cs].

141. D. Szklarczyk, A. L. Gable, D. Lyon, A. Junge, S. Wyder, J. Huerta-Cepas, M. Simonovic, N. T. Doncheva, J. H. Morris, P. Bork, L. J. Jensen, and C. v.

Mering. STRING v11: protein-protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets. *Nucleic Acids Research*, 47(D1):D607–D613, Jan. 2019. ISSN 1362-4962. doi: 10.1093/nar/gky1131.

142. N. Tishby and N. Zaslavsky. Deep learning and the information bottleneck principle. In *2015 IEEE Information Theory Workshop (ITW)*, pages 1–5, Apr. 2015. doi: 10.1109/ITW.2015.7133169.

143. N. Tishby, F. C. Pereira, and W. Bialek. The information bottleneck method, Apr. 2000. URL `http://arxiv.org/abs/physics/0004057`. arXiv:physics/0004057.

144. J. Topping, F. Di Giovanni, B. P. Chamberlain, X. Dong, and M. M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. *arXiv:2111.14522 [cs, stat]*, Nov. 2021. URL `http://arxiv.org/abs/2111.14522`. arXiv: 2111.14522.

145. J. Topping, F. Di Giovanni, B. P. Chamberlain, X. Dong, and M. M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. *arXiv:2111.14522 [cs, stat]*, Mar. 2022. URL `http://arxiv.org/abs/2111.14522`. arXiv: 2111.14522.

146. H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom. Llama 2: Open Foundation and Fine-Tuned Chat Models, July 2023. URL `http://arxiv.org/abs/2307.09288`. arXiv:2307.09288 [cs].

147. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://papers.nips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html`.

148. P. Veličković. Everything is connected: Graph neural networks. *Current Opinion in Structural Biology*, 79:102538, Apr. 2023. ISSN 0959-440X. doi: 10.1016/j.sbi.2023.102538. URL `https://www.sciencedirect.com/science/article/pii/S0959440X2300012X`.

149. P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph Attention Networks. *arXiv:1710.10903 [cs, stat]*, Feb. 2018. URL http://arxiv.org/abs/1710.10903. arXiv: 1710.10903.

150. C. Von Mering, R. Krause, B. Snel, M. Cornell, S. G. Oliver, S. Fields, and P. Bork. Comparative assessment of large-scale data sets of protein–protein interactions. *Nature*, 417(6887):399–403, 2002. Publisher: Nature Publishing Group.

151. K. Wang, V. Muthukumar, and C. Thrampoulidis. Benign Overfitting in Multiclass Classification: All Roads Lead to Interpolation, 2023. _eprint: 2106.10865.

152. L. Wang, M. Zhang, Z. Jia, Q. Li, C. Bao, K. Ma, J. Zhu, and Y. Zhong. AFEC: Active Forgetting of Negative Transfer in Continual Learning, Nov. 2021. URL http://arxiv.org/abs/2110.12187. arXiv:2110.12187 [cs].

153. L. Wang, L. Li, D. Dai, D. Chen, H. Zhou, F. Meng, J. Zhou, and X. Sun. Label Words are Anchors: An Information Flow Perspective for Understanding In-Context Learning, Dec. 2023. URL http://arxiv.org/abs/2305.14160. arXiv:2305.14160 [cs].

154. X. Wang and M. Zhang. How Powerful are Spectral Graph Neural Networks. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 23341–23362. PMLR, July 2022. URL https://proceedings.mlr.press/v162/wang22am.html.

155. X. Wang, H. Yang, and M. Zhang. Neural Common Neighbor with Completion for Link Prediction, Feb. 2023. URL http://arxiv.org/abs/2302.00890. arXiv:2302.00890 [cs].

156. Y. Wang, W. Wang, Y. Liang, Y. Cai, J. Liu, and B. Hooi. NodeAug: Semi-Supervised Node Classification with Data Augmentation. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pages 207–217, New York, NY, USA, Aug. 2020. Association for Computing Machinery. ISBN 978-1-4503-7998-4. doi: 10.1145/3394486.3403063. URL https://doi.org/10.1145/3394486.3403063.

157. Z. Wang, Y. Zhou, L. Hong, Y. Zou, H. Su, and S. Chen. Pairwise Learning for Neural Link Prediction. *arXiv:2112.02936 [cs]*, Jan. 2022. URL http://arxiv.org/abs/2112.02936. arXiv: 2112.02936.

158. D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998. URL https://api.semanticscholar.org/CorpusID:3034643.

159. J. Wei, J. Wei, Y. Tay, D. Tran, A. Webson, Y. Lu, X. Chen, H. Liu, D. Huang, D. Zhou, and T. Ma. Larger language models do in-context learning differently, Mar. 2023. URL http://arxiv.org/abs/2303.03846. arXiv:2303.03846 [cs].

160. B. Weisfeiler and A. Leman. The reduction of a graph to canonical form and the algebra which appears therein. *NTI, Series*, 2(9):12–16, 1968.

161. F. Wu, T. Zhang, A. H. d. Souza Jr., C. Fifty, T. Yu, and K. Q. Weinberger. Simplifying Graph Convolutional Networks. *arXiv:1902.07153 [cs, stat]*, June 2019. URL http://arxiv.org/abs/1902.07153. arXiv: 1902.07153.

162. Q. Wu, H. Zhang, J. Yan, and D. Wipf. Handling Distribution Shifts on Graphs: An Invariance Perspective. May 2022. URL https://openreview.net/forum?id=FQOC5u-1egI.

163. T. Wu, H. Ren, P. Li, and J. Leskovec. Graph Information Bottleneck, Oct. 2020. URL http://arxiv.org/abs/2010.12811. arXiv:2010.12811 [cs, stat].

164. Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, Oct. 2016. URL http://arxiv.org/abs/1609.08144. arXiv:1609.08144 [cs].

165. Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, Jan. 2021. ISSN 2162-237X, 2162-2388. doi: 10.1109/TNNLS.2020.2978386. URL http://arxiv.org/abs/1901.00596. arXiv: 1901.00596.

166. Q. Xie, Z. Dai, E. Hovy, M.-T. Luong, and Q. V. Le. Unsupervised Data Augmentation for Consistency Training, Nov. 2020. URL http://arxiv.org/abs/1904.12848. arXiv:1904.12848 [cs, stat].

167. K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How Powerful are Graph Neural Networks? *CoRR*, abs/1810.00826, 2018. URL http://arxiv.org/abs/1810.00826. arXiv: 1810.00826.

168. Y. Yang, R. N. Lichtenwalter, and N. V. Chawla. Evaluating link prediction methods. *Knowledge and Information Systems*, 45(3):751–782, Dec. 2015. ISSN 0219-3116. doi: 10.1007/s10115-014-0789-0. URL https://doi.org/10.1007/s10115-014-0789-0.

169. Z. Yang, W. Cohen, and R. Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pages 40–48. PMLR, 2016.

170. G. Yehudai, E. Fetaya, E. Meirom, G. Chechik, and H. Maron. From Local Structures to Size Generalization in Graph Neural Networks, July 2021. URL http://arxiv.org/abs/2010.08853. arXiv:2010.08853 [cs, stat].

171. H. Yin, M. Zhang, Y. Wang, J. Wang, and P. Li. Algorithm and System Co-design for Efficient Subgraph-based Graph Representation Learning. *Proceedings of the VLDB Endowment*, 15(11):2788–2796, July 2022. ISSN 2150-8097. doi: 10.14778/3551793.3551831. URL http://arxiv.org/abs/2202.13538. arXiv:2202.13538 [cs].

172. R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pages 974–983, New York, NY, USA, July 2018. Association for Computing Machinery. ISBN 978-1-4503-5552-0. doi: 10.1145/3219819.3219890. URL https://doi.org/10.1145/3219819.3219890.

173. J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 3320–3328, Cambridge, MA, USA, Dec. 2014. MIT Press.

174. Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen. Graph Contrastive Learning with Augmentations, Apr. 2021. URL http://arxiv.org/abs/2010.13902. arXiv:2010.13902 [cs].

175. J. Yu, T. Xu, Y. Rong, Y. Bian, J. Huang, and R. He. Graph Information Bottleneck for Subgraph Recognition, Oct. 2020. URL http://arxiv.org/abs/2010.05563. arXiv:2010.05563 [cs, stat].

176. S. Yun, S. Kim, J. Lee, J. Kang, and H. J. Kim. Neo-GNNs: Neighborhood Overlap-aware Graph Neural Networks for Link Prediction. Nov. 2021. URL https://openreview.net/forum?id=Ic9vRN3VpZ.

177. M. Zhang and Y. Chen. Weisfeiler-Lehman Neural Machine for Link Prediction. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 575–583, Halifax NS Canada, Aug. 2017. ACM. ISBN 978-1-4503-4887-4. doi: 10.1145/3097983.3097996. URL https://dl.acm.org/doi/10.1145/3097983.3097996.

178. M. Zhang and Y. Chen. Link Prediction Based on Graph Neural Networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/53f0d7c537d99b3824f0f99d62ea2428-Paper.pdf.

179. M. Zhang and P. Li. Nested Graph Neural Networks, 2021. URL https://arxiv.org/abs/2110.13197.

180. M. Zhang, Z. Cui, S. Jiang, and Y. Chen. Beyond link prediction: Predicting hyperlinks in adjacency space. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

181. M. Zhang, Z. Cui, M. Neumann, and Y. Chen. An End-to-End Deep Learning Architecture for Graph Classification. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018. ISSN 2374-3468. doi: 10.1609/aaai.v32i1.11782. URL https://ojs.aaai.org/index.php/AAAI/article/view/11782. Number: 1.

182. M. Zhang, P. Li, Y. Xia, K. Wang, and L. Jin. Labeling Trick: A Theory of Using Graph Neural Networks for Multi-Node Representation Learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 9061–9073. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/4be49c79f233b4f4070794825c323733-Paper.pdf.

183. S. Zhang, H. Chen, X. Sun, Y. Li, and G. Xu. Unsupervised Graph Poisoning Attack via Contrastive Loss Back-propagation. In *Proceedings of the ACM Web Conference 2022*, pages 1322–1330, Apr. 2022. doi: 10.1145/3485447.3512179. URL http://arxiv.org/abs/2201.07986. arXiv:2201.07986 [cs].

184. X. Zhang, J. Zhao, and Y. LeCun. Character-level Convolutional Networks for Text Classification, Apr. 2016. URL http://arxiv.org/abs/1509.01626. arXiv:1509.01626 [cs].

185. T. Zhao, G. Liu, D. Wang, W. Yu, and M. Jiang. Learning from Counterfactual Links for Link Prediction. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 26911–26926. PMLR, July 2022. URL https://proceedings.mlr.press/v162/zhao22e.html.

186. T. Zhao, W. Jin, Y. Liu, Y. Wang, G. Liu, S. Günnemann, N. Shah, and M. Jiang. Graph Data Augmentation for Graph Machine Learning: A Survey, Jan. 2023. URL http://arxiv.org/abs/2202.08871. arXiv:2202.08871 [cs].

187. C. Zheng, B. Zong, W. Cheng, D. Song, J. Ni, W. Yu, H. Chen, and W. Wang. Robust Graph Representation Learning via Neural Sparsification. In *Proceedings of the 37th International Conference on Machine Learning*, pages 11458–11468. PMLR, Nov. 2020. URL https://proceedings.mlr.press/v119/zheng20d.html. ISSN: 2640-3498.

188. Q. Zheng, X. Xia, K. Zhang, E. Kharlamov, and Y. Dong. On the distribution alignment of propagation in graph neural networks. *AI Open*, 3:218–228, Jan. 2022. ISSN 2666-6510. doi: 10.1016/j.aiopen.2022.11.006. URL https://www.sciencedirect.com/science/article/pii/S2666651022000213.

189. Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang. Random Erasing Data Augmentation, Nov. 2017. URL http://arxiv.org/abs/1708.04896. arXiv:1708.04896 [cs].

190. H. Zhou, J. Lan, R. Liu, and J. Yosinski. Deconstructing Lottery Tickets: Zeros, Signs, and the Supermask. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/hash/1113d7a76ffceca1bb350bfe145467c6-Abstract.html.

191. T. Zhou, L. Lü, and Y.-C. Zhang. Predicting missing links via local information. *The European Physical Journal B*, 71(4):623–630, 2009. Publisher: Springer.

192. Y. Zhou, G. Kutyniok, and B. Ribeiro. OOD Link Prediction Generalization Capabilities of Message-Passing GNNs in Larger Test Graphs, 2022. URL https://arxiv.org/abs/2205.15117.

193. Q. Zhu, N. Ponomareva, J. Han, and B. Perozzi. Shift-Robust GNNs: Overcoming the Limitations of Localized Graph Training data. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 27965–27977. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/eb55e369affa90f77dd7dc9e2cd33b16-Paper.pdf.

194. X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using Gaussian fields and harmonic functions. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, pages 912–919, Washington, DC, USA, Aug. 2003. AAAI Press. ISBN 978-1-57735-189-4.

195. Z. Zhu, Z. Zhang, L.-P. Xhonneux, and J. Tang. Neural bellman-ford networks: A general graph neural network framework for link prediction. *Advances in Neural Information Processing Systems*, 34, 2021.

196. D. Zügner, A. Akbarnejad, and S. Günnemann. Adversarial Attacks on Neural Networks for Graph Data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pages 2847–2856, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 978-1-4503-5552-0. doi: 10.1145/3219819.3220078. URL https://doi-org.proxy.library.nd.edu/10.1145/3219819.3220078. event-place: London, United Kingdom.