LINK PREDICTION REVISITED: FROM EVALUATION PITFALLS TO LANGUAGE MODEL SYNERGIES

By

Juanhui Li

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computer Science—Doctor of Philosophy

2025

ABSTRACT

Artificial intelligence (AI) and machine learning (ML) have significantly impacted many aspects of daily life, with numerous methods involving structural graph data. Graphs, which model relationships between different entities, are widely used to represent real-world data, including social networks, transportation systems, chemical molecules, power and communication networks, and user-item interactions in recommendation systems. A fundamental task in graph data analysis is link prediction, which predicts connections between entities and is essential for understanding their relationships. For example, link prediction allows us to determine whether two individuals are friends in a social network or if a user will purchase an item in a recommendation system. Consequently, link prediction is crucial for advancing graph-based applications in real-world ML applications. However, several challenges impede progress in this area. Specifically, 1) existing evaluation settings are not unified or rigorous, leading to inconsistent and sometimes suboptimal results, and 2) graph nodes are frequently associated with textual attributes containing rich semantic information, and this data has become increasingly abundant. Language models excel at processing textual data to capture semantic insights. However, effectively integrating textual information with graph data to enhance real-world applications remains under-explored.

In light of these challenges, this dissertation seeks to advance link prediction from two main perspectives: 1) identifying evaluation pitfalls across various graph types to inspire more advanced methods for link prediction, and 2) leveraging language models in synergy with link prediction techniques to enhance a range of real-world applications. From the first perspective, we investigate evaluation pitfalls in both uni-relational and multi-relational graphs. Based on these findings, we propose methods to mitigate the identified issues or develop more effective and efficient models. From the second perspective, we explore the use of language models to improve link prediction in recommendation systems, and conversely, apply link prediction techniques to enhance language models in query understanding tasks. By working on these two perspectives, this dissertation not only contributes to the development of more robust link prediction methods but also facilitates their application in practical, real-world scenarios.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my advisor, Dr. Jiliang Tang, for his unwavering support and insightful guidance throughout my Ph.D. journey. Working with Dr. Tang has been a transformative experience. Under his mentorship, I have grown into an independent researcher, learning how to formulate meaningful research questions, communicate ideas effectively, and maintain a rigorous and curious mindset. Beyond academia, Dr. Tang has been a source of inspiration through his integrity and dedication to his students. His ability to lead with compassion and clarity has left a lasting impact on me, and his trust in my potential has pushed me to achieve far more than I ever thought possible. I feel incredibly fortunate to have had him as my advisor, and I will always carry his lessons forward in my career and life. I would extend my gratitude to my other Ph.D. committee members: Dr. Hui Liu, Dr. Yue Xing, Dr. Jun Wu, Dr. Xianfeng Tang, for their insightful comments and helpful suggestions.

I am grateful to have had the pleasure and fortune of having supportive and encouraging colleagues during my Ph.D. I am thankful to all my colleagues from the Data Science and Engineering Lab: , Dr. Yao Ma, Dr. Jamell Dacon, Dr. Wenqi Fan, Dr. Tyler Derr, Yaxin Li, Dr. Haochen Liu, Dr. Hua Liu, Dr. Xiaorui Liu, Dr. Wei Jin, Dr. Wentao Wang, Dr. Xiaoyang Wang, Dr. Yiqi Wang, Dr. Zhiwei Wang, Dr. Han Xu, Dr. Xiangyu Zhao, Harry Shomer, Dr. Hongzhi Wen, Pengfei He, Haitao Mao, Jiayuan Ding, Haoyu Han, Yingqian Cui, Jie Ren, Kaiqi Yang, Hang Li, Yuxuan Wan, Zhikai Chen, Kai Guo, Wenzhuo Tang, Dr. Remy Liu, Yuping Lin, Jay Revolinsky, Yu Song, Jingzhe Liu, Xinnan Dai, Hanbing Wang, Shenglai Zeng, Li Ma, Shen Dong, Quang Truong, Bingheng Li, Bo Wang, Yucheng Chu. I am also thankful for the collaboration from outside the DSE lab: Dr. Amin Javari at the Home Depot; Dr. Dawei Yin, Dr. Wei Zeng, Dr. Suqi Cheng, Dr. Shuaiqiang Wang at the Baidu Inc.; Dr. Neil Shah at the Snap; Sreyashi Nag, Dr. Hui Liu, Dr. Xianfeng Tang, Dr. Sheikh Sarwar, Dr. Limeng Cui, Dr. Hansu Gu Dr. Qi He at Amazon, Dr. Suhang Wang at The Pennsylvania State University.

Finally, I would again like to express my gratitude to my family, my dear husband and friends, and my lovely cats, for their unconditional love and support.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Motivation

Graph is a universal data structure used to model relationships between different entities. Due to its ubiquitous nature, numerous artificial intelligence and machine learning methods have been developed to leverage structural graph data. Graphs are widely used to represent real-world applications such as social networks [95], transportation infrastructures [87], chemical molecules [120], and recommendation systems [125]. A fundamental task in graph data analysis is link prediction [62, 74, 101], which aims to predict potential connections between entities and is critical for understanding the underlying relationships. Link prediction plays a central role in many real-world applications. For example, recommending friends in social networks [95] or predicting whether a user will purchase an item in recommendation systems [125]. As such, link prediction is essential for the development of graph-based machine learning applications. To handle this task, a large body of work has focused on graph neural networks (GNNs) [128], which have demonstrated strong performance in learning from graph-structured data.

However, several challenges hinder progress in this area. Specifically: 1. evaluation settings are often neither unified nor rigorous, resulting in inconsistent and sometimes suboptimal findings. Since model performance is assessed and interpreted based on these evaluations, rigorous evaluation protocols are essential for accurate analysis. Nevertheless, existing literature [117, 146, 110, 15] primarily focuses on designing novel model architectures to address various challenges, while issues related to evaluation settings are largely overlooked. For example, due to non-rigorous evaluation, we find that a commonly believed key component has limited impact in knowledge graph completion tasks. 2. Graph nodes are often associated with rich textual attributes, and such information is increasingly abundant. For instance, users in social networks typically have profiles, and items in e-commerce systems are described by titles and descriptions. This textual information is particularly valuable for nodes with sparse connectivity, which constitute a large portion of real-world graphs [72], as it offers complementary semantic context beyond the structural information provided by the graph. Language models [50] excel at capturing semantic representations from text. However, integrating textual information with graph data to improve performance in practical applications remains under-explored.

To address these challenges, we aim to advance link prediction from two main perspectives. First, we identify evaluation pitfalls across various graph types to advance the development of more rigorous and effective link prediction methods. Our analysis spans both uni-relational and multi-relational graphs, uncovering several unexpected findings. These insights can inform the design of more robust models and help align link prediction techniques more closely with real-world applications. Second, we investigate the integration of language models with link prediction techniques to enhance practical performance across diverse domains. Specifically, we explore the use of language models to improve link prediction in recommendation systems, as well as the use of link prediction to support language models in query understanding tasks. Through working on these two lines, our work offers new perspectives and insights that can guide the development of more effective link prediction methods and facilitation their applicability in real-world scenarios.

1.2 Contributions

We summarize the major contributions of this proposal as follows:

- In chapter 2, we identify several evaluation pitfalls in uni-relational graph [62] settings and developed a comprehensive benchmark spanning a wide range of datasets and methods. Based on several limitations observed in existing evaluation procedures, I proposed a more practical evaluation framework called HeaRT (Heuristic Related Sampling Technique). By enabling a more rigorous and realistic assessment, HeaRT provides a foundation for guiding the field towards developing more effective models, thereby advancing the state of the art in link prediction.
- In chapter 3, we surprisingly find that MLP-based models achieve competitive performance compared to three MPNN-based models across various datasets [61]. Additionally, scoring and loss functions were identified as having a significant impact on performance. Building on these insights, I developed an ensemble-based MLP model that not only delivers stronger performance but also offers greater computational efficiency.

- In chapter 4, we identify an imbalance issue in the traditional framework for integrating graph and text information [59]. To address this challenge, I propose a novel approach, AlterRec, which employs an alternative training strategy. This strategy enables implicit interactions between ID and textual features, fostering mutual learning and significantly enhancing overall performance.
- In chapter 5, we propose a pre-training framework, GE-BERT [63], designed to preserve both semantic information and query graph structure. This framework is further fine-tuned for query understanding tasks. Extensive experiments on both offline and online benchmarks demonstrate the superior performance of GE-BERT compared to existing methods.

CHAPTER 2

EVALUATION PITFALLS IN UNI-RELATIONAL GRAPH

Link prediction attempts to predict whether an unseen edge exists based on only a portion of edges of a graph. A flurry of methods have been introduced in recent years that attempt to make use of graph neural networks (GNNs) for this task. Furthermore, new and diverse datasets have also been created to better evaluate the effectiveness of these new models. However, multiple pitfalls currently exist that hinder our ability to properly evaluate these new methods. These pitfalls mainly include: (1) Lower than actual performance on multiple baselines, (2) A lack of a unified data split and evaluation metric on some datasets, and (3) An unrealistic evaluation setting that uses easy negative samples. To overcome these challenges, we first conduct a fair comparison across prominent methods and datasets, utilizing the same dataset and hyperparameter search settings. We then create a more practical evaluation setting based on a **He**uristic **R**elated Sampling **T**echnique (HeaRT), which samples hard negative samples via multiple heuristics. The new evaluation setting helps promote new challenges and opportunities in link prediction by aligning the evaluation with real-world situations.

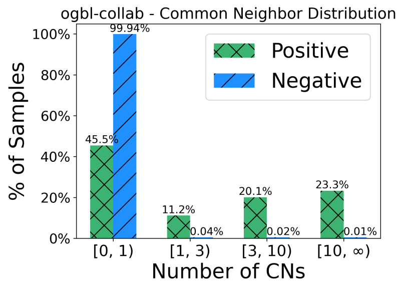
2.1 Introduction

The task of link prediction is to determine the existence of an edge between two unconnected nodes in a graph. Existing link prediction algorithms attempt to estimate the proximity of different pairs of nodes in the graph, where node pairs with a higher proximity are more likely to interact [71]. Link prediction is applied in many different domains including social networks [17], biological networks [55], and recommender systems [41].

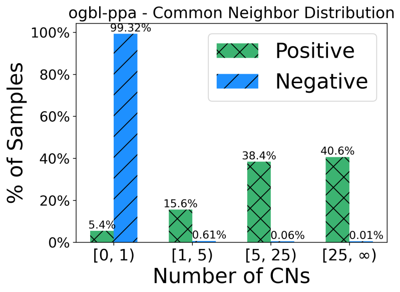
Graph neural networks (GNNs) [128] have gained prominence in recent years with many new frameworks being proposed for a variety of different tasks. Corresponding to the rise in popularity of GNNs, there has been a number of studies that attempt to critically examine the effectiveness of different GNNs on various tasks. This can be seen for the task of node classification [98], graph classification [27], knowledge graph completion (KGC) [93, 2, 105], and others [25].

However, despite a number of new GNN-based methods being proposed [145, 15, 143, 117] for link prediction, there is currently no work that attempts to carefully examine recent advances in link prediction methods. Upon examination, we find that there are several pitfalls in regard to model evaluation that impede our ability to properly evaluate current methods. This includes:

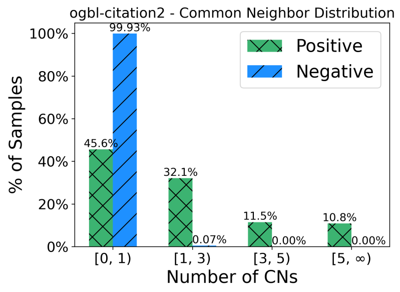
1. **Lower than Actual Performance**. We observe that the current performance of multiple models is underreported. For some methods, such as standard GNNs, this is due to poor hyperparameter tuning. Once properly tuned, they can even achieve the best overall performance on some metrics (see SAGE [33] in Table 2.1). Furthermore, for other methods like Neo-GNN [143] we can achieve around an 8.5 point increase in Hits@50 on ogbl-collab relative to the originally reported performance. This results in Neo-GNN achieving the best overall performance on ogbl-collab in our study (see Table 2.2). Such problems obscure the true performance of different models, making it difficult to draw reliable conclusions from the current results.
2. **Lack of Unified Settings**. For Cora, Citeseer, and Pubmed datasets [135], there exists no unified data split and evaluation metrics used for each individually. For the data split, some works [112, 157] use a single fixed train/valid/test split with percentages 85/5/10%. More recent works [15, 117] use 10 random splits of size 70/10/20%. In terms of the evaluation metrics, some studies [15, 117] use ranking-based metrics such as MRR or Hits@K while others [52, 157] report the area under the curve (AUC). This is despite multiple studies that argue that AUC is a poor metric for evaluating link prediction [133, 42]. Additionally, for both the planetoid (i.e., Cora, Citeseer and Pubmed) and ogbl-collab datasets, some methods incorporate the validation edges during testing [15, 39], while others [143, 117] don't. This lack of a unified setting makes it difficult to draw a comparison and hampers our ability to determine which methods perform best on these datasets.
3. **Unrealistic Evaluation Setting**. During the evaluation, we are given a set of true samples (i.e., positive samples) and a set of false samples (i.e., negative samples). We are tasked with learning a classifier f that assigns a higher probability to the positive samples than the negatives. The current evaluation setting uses the same set of randomly selected negative samples for each positive sample. We identify two potential problems with the current evaluation procedure. **(1)** It



(a) ogbl-collab



(b) ogbl-ppa



(c) ogbl-citation2

Figure 2.1 Common neighbor distribution for the positive and negative test samples for the ogbl-collab, ogbl-ppa, and ogbl-citation2 datasets under the existing evaluation setting.

is not aligned with real-world settings. In a real-world scenario, we typically care about predicting links for a specific node. For example, in friend recommendations, we aim to recommend friends for a specific user u. To evaluate such models for u, we strive to rank node pairs including u. However, this does not hold in the current setting as u is not included in most of the negative samples. **(2)** The current evaluation setting makes the task too easy. As such, it may not reflect the model performance in real-world applications. This is because the nodes in a randomly selected negative "node pair" are likely to be unrelated to each other. As shown in Figure 2.1, almost all negative samples in the test data have no common neighbors, a typically strong heuristic, making them trivial to classify them.

To account for these issues, we propose to first conduct a fair and reproducible evaluation among current link prediction methods under the existing evaluation setting. We then design a new evaluation strategy that is more aligned with a real-world setting and detail our results. Our key contributions are summarized below:

1. **Reproducible and Fair Comparison**. We conduct a fair comparison of different models across multiple common datasets. To ensure a fair comparison, we tune all models on the same set of hyperparameters. We further evaluate different models using multiple types of evaluation metrics. For the Planetoid datasets [135], we further use a unified data split to facilitate a point of comparison between models. To the best of our knowledge, there are no recent efforts to comprehensively benchmark link prediction methods (several exist for KGC [105, 2, 93]).

Furthermore, we open-source the implementation in our analysis to enable others in their analyses.

2. **New Evaluation Setting**. We recognize that the current negative sampling strategy used in evaluation is unrealistic and easy. To counter these issues, we first use a more realistic setting of tailoring the negatives to each positive sample. This is achieved by restricting them to be corruptions of the positive sample (i.e., containing one of its two nodes as defined in Eq. (2.3)). Given the prohibitive cost of utilizing all possible corruptions, we opt instead to only rank against K negatives for each positive sample. In order to choose the most relevant and difficult corruptions, we propose a **He**uristic **R**elated Sampling **T**echnique (HeaRT), which selects them based on a combination of multiple heuristics. This creates a more challenging task than the previous evaluation strategy and allows us to better assess the capabilities of current methods.

The rest of the paper is structured as follows. In Section 2.2 we introduce the models, datasets, and settings used for conducting a fair comparison between methods. In Section 2.3 we show the results of the fair comparison under the existing evaluation setting and discuss our main observations. Lastly, in Section 2.4 we introduce our new evaluation setting. We then detail and discuss the performance of different methods using our new setting.

2.2 Preliminaries

2.2.1 Task Formulation

In this section we formally define the task of link prediction. Let $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ be a graph where $\mathcal{V}$ and $\mathcal{E}$ are the set of nodes and edges in the graph, respectively. Furthermore, let $X \in \mathcal{R}^{|V| \times d}$ be a set of d-dimensional features for each node. Link prediction aims to predict the likelihood of a link existing between two nodes given the structural and feature information. For a pair of nodes u and v, the probability of a link existing, $p(u, v)$, is therefore given by:

$$p(u, v) = p(u, v \mid \mathcal{G}, X). \tag{2.1}$$

Traditionally, $p(u, v)$ was estimated via non-learnable heuristic methods [76, 66]. More recently, methods that use learnable parameters have gained popularity [145, 15]. These methods attempt to estimate $p(u, v)$ via a learnable function f such that:

$$p(u, v) = f(u, v \mid \mathcal{G}, X, \Theta), \tag{2.2}$$

where Θ represents a set of learnable parameters. A common choice of f are graph neural networks [53]. In the next subsection we detail the various link prediction methods used in this study.

2.2.2 Link Prediction Methods

In this section we given an overview of the different methods used in this study. Conventional methods [76, 66] often exploit hand-craft graph structural properties (i.e., heuristics) between node pairs. GNNs attempt to learn the structural information to facilitate link prediction [146, 117, 15]. Given the strong performance of pairwise-based heuristics [143, 117], some recent works use both GNNs and pairwise information, demonstrating strong performance.

For our study, we consider both traditional and state-of-the-art GNN-based models. They can be roughly organized into four categories. **1) Heuristic methods**: Common Neighbor (CN) [80], Adamic Adar (AA) [1], Resource Allocation (RA) [155], Shortest Path [66], and Katz [49]. These methods define a score to indicate the link existence based on the graph structure. Among them, CN, AA, and RA are based on the common neighbors, while Shortest Path and Katz are based on the path information. **2) Embedding methods**: Matrix factorization (MF) [76], Multilayer Perceptron (MLP) and Node2Vec [31]. These methods are trained to learn low-dimensional node embeddings that are used to predict the likelihood of node pairs existing. **3) GNN methods**: GCN [137], GAT [112], SAGE [33], and GAE [52]. These methods attempt to integrate the multi-hop graph structure based on the message passing paradigm. **4) GNN + Pairwise Information methods**: Standard GNN methods, while powerful, are not able to capture link-specific information [146]. As such, works have been proposed that augment GNN methods by including additional information to better capture the relation between the nodes in the link we are predicting. SEAL [146], BUDDY [15], and NBFNet [157] use the subgraph features. Neo-GNN [143], NCN [117], and NCNC [117] are based on common neighbor information. Lastly, PEG [114] uses the positional encoding derived from the graph structure.

2.2.3 Datasets and Experimental Settings

In this section we summarize the datasets and evaluation and training settings. We note that the settings depend on the specific dataset. More details are given in Appendix A.3.

Datasets. We limit our experiments to homogeneous graphs, which are the most commonly used datasets for link prediction. This includes the small-scale datasets, i.e., Cora, Citeseer, Pubmed [135], and large-scale datasets in the OGB benchmark [39], i.e., ogbl-collab, ogbl-ddi, ogbl-ppa, and ogbl-citation2. We summarize the statistics and split ratio of each dataset in Appendix A.3.

Metrics. For evaluation, we use both the area under the curve (AUC) and ranking-based metrics, i.e., mean reciprocal rank (MRR) and Hits@K. For Cora, Citeseer, and Pubmed we adopt $K \in \{1, 3, 10, 100\}$. We note that $K = 100$ is reported in some recent works [15, 117]). However due to the small number of negatives used during evaluation (e.g., ≈ 500 for Cora and Citeseer) $K = 100$ is likely not informative. For the OGB datasets, we adopt $K \in \{20, 50, 100\}$ to keep consistent with the original study [39]. Please see Appendix A.2.1 for the formal definitions of the various evaluation metrics.

Hyperparameter Ranges. We conduct a grid hyperparameter search across a comprehensive range of values. For Cora, Citeseer, and Pubmed this includes: learning rate (0.01, 0.001), dropout (0.1, 0.3, 0.5), weight decay (1e-4, 1e-7, 0), number of model layers (1, 2, 3), number of prediction layers (1, 2, 3), and the embedding size (128, 256). Due to the large size of the OGB datasets, it's infeasible to tune over such a large range. Therefore, following the most commonly used settings among published hyperparameters, we fix the weight decay to 0, the number of model and prediction layers to be 3, and the embedding size to be 256. The best hyperparameters are chosen based on the validation performance. We note that several exceptions exist to these ranges when they result in significant performance degradations (see Appendix A.3 for more details). We further follow the existing setting and only sample one negative sample per positive sample during training.

Existing Evaluation Settings. In the evaluation stage, the same set of randomly sampled negatives are used for all positive samples. We note that one exception is ogbl-citation2, where they randomly sample 1000 negative samples per positive sample. For Cora, Citeseer, and Pubmed the

Table 2.1 Results on Cora, Citeseer, and Pubmed(%) under the existing evaluation setting. Highlighted are the results ranked first (green), second (blue), and third (orange).

	Models	Cora		Citeseer		Pubmed	
		MRR	AUC	MRR	AUC	MRR	AUC
Heuristic	CN	20.99	70.85	28.34	67.49	14.02	63.9
	AA	31.87	70.96	29.37	67.49	16.66	63.9
	RA	30.79	70.96	27.61	67.48	15.63	63.9
	Shortest Path	12.45	81.08	31.82	75.5	7.15	74.64
	Katz	27.4	81.17	38.16	75.37	21.44	74.86
Embedding	Node2Vec	**37.29 ± 8.82**	90.97 ± 0.64	44.33 ± 8.99	94.46 ± 0.59	34.61 ± 2.48	93.14 ± 0.18
	MF	14.29 ± 5.79	80.29 ± 2.26	24.80 ± 4.71	75.92 ± 3.25	19.29 ± 6.29	93.06 ± 0.43
	MLP	31.21 ± 7.90	95.32 ± 0.37	43.53 ± 7.26	94.45 ± 0.32	16.52 ± 4.14	98.34 ± 0.10
GNN	GCN	32.50 ± 6.87	95.01 ± 0.32	50.01 ± 6.04	95.89 ± 0.26	19.94 ± 4.24	98.69 ± 0.06
	GAT	31.86 ± 6.08	93.90 ± 0.32	48.69 ± 7.53	96.25 ± 0.20	18.63 ± 7.75	98.20 ± 0.07
	SAGE	**37.83 ± 7.75**	**95.63 ± 0.27**	47.84 ± 6.39	**97.39 ± 0.15**	22.74 ± 5.47	**98.87 ± 0.04**
	GAE	29.98 ± 3.21	95.08 ± 0.33	**63.33 ± 3.14**	**97.06 ± 0.22**	16.67 ± 0.19	97.47 ± 0.08
GNN+Pairwise Info	SEAL	26.69 ± 5.89	90.59 ± 0.75	39.36 ± 4.99	88.52 ± 1.40	**38.06 ± 5.18**	97.77 ± 0.40
	BUDDY	26.40 ± 4.40	95.06 ± 0.36	**59.48 ± 8.96**	96.72 ± 0.26	23.98 ± 5.11	98.2 ± 0.05
	Neo-GNN	22.65 ± 2.60	93.73 ± 0.36	53.97 ± 5.88	94.89 ± 0.60	31.45 ± 3.17	98.71 ± 0.05
	NCN	32.93 ± 3.80	**96.76 ± 0.18**	54.97 ± 6.03	97.04 ± 0.26	**35.65 ± 4.60**	**98.98 ± 0.04**
	NCNC	29.01 ± 3.83	**96.90 ± 0.28**	**64.03 ± 3.67**	**97.65 ± 0.30**	25.70 ± 4.48	**99.14 ± 0.03**
	NBFNet	**37.69 ± 3.97**	92.85 ± 0.17	38.17 ± 3.06	91.06 ± 0.15	**44.73 ± 2.12**	98.34 ± 0.02
	PEG	22.76 ± 1.84	94.46 ± 0.34	56.12 ± 6.62	96.15 ± 0.41	21.05 ± 2.85	96.97 ± 0.39

number of negative samples is equal to the number of positive samples. For the OGB datasets, we use the existing fixed set of randomly chosen negatives found in [39]. Furthermore, for ogbl-collab we follow the existing protocol [39] and include the validation edges in the training graph during testing. This setting is adopted on ogbl-collab under both the existing and new evaluation setting.

2.3 Fair Comparison Under the Existing Setting

In this section, we conduct a fair comparison among link prediction methods. This comparison is spurred by the multiple pitfalls noted in Section 2.1, which include lower-than-actual model performance, multiple data splits, and inconsistent evaluation metrics. These pitfalls hinder our ability to fairly compare different methods. To rectify this, we conduct a fair comparison adhering to the settings listed in section 2.2.3.

The results are split into two tables. The results for Cora, Citeseer, and Pubmed are shown in Table 2.1 and OGB in Table 2.2. For simplicity, we only present the AUC and MRR for Cora, Citeseer, and Pubmed. For OGB datasets, we include AUC and the original ranking metric reported in [39] to allow a convenient comparison (Hits@20 for ogbl-ddi, Hits@50 for ogbl-collab, Hits@100

Table 2.2 Results on OGB datasets (%) under the existing evaluation setting. Highlighted are the results ranked first (green), second (blue), and third (orange).

	Models	ogbl-collab		ogbl-ddi		ogbl-ppa		ogbl-citation2
		Hits@50	AUC	Hits@20	AUC	Hits@100	AUC	MRR
Heuristic	CN	61.37	82.78	17.73	95.2	27.65	97.22	74.3
	AA	64.17	82.78	18.61	95.43	32.45	97.23	75.96
	RA	63.81	82.78	6.23	96.51	**49.33**	97.24	76.04
	Shortest Path	46.49	96.51	0	59.07	0	99.13	>24h
	Katz	64.33	90.54	17.73	95.2	27.65	97.22	74.3
Embedding	Node2Vec	49.06 ± 1.04	96.24 ± 0.15	34.69 ± 2.90	99.78 ± 0.04	26.24 ± 0.96	99.77 ± 0.00	45.04 ± 0.10
	MF	41.81 ± 1.67	83.75 ± 1.77	23.50 ± 5.35	99.46 ± 0.10	28.4 ± 4.62	99.46 ± 0.10	50.57 ± 12.14
	MLP	35.81 ± 1.08	95.91 ± 0.08	N/A	N/A	0.45 ± 0.04	90.23 ± 0.00	38.07 ± 0.09
GNN	GCN	54.96 ± 3.18	97.89 ± 0.06	**49.90 ± 7.23**	99.86 ± 0.03	29.57 ± 2.90	**99.84 ± 0.03**	84.85 ± 0.07
	GAT	55.00 ± 3.28	97.11 ± 0.09	31.88 ± 8.83	99.63 ± 0.21	OOM	OOM	OOM
	SAGE	59.44 ± 1.37	**98.08 ± 0.03**	49.84 ± 15.56	**99.96 ± 0.00**	41.02 ± 1.94	99.82 ± 0.00	83.06 ± 0.09
	GAE	OOM	OOM	7.09 ± 6.02	75.34 ±15.96	OOM	OOM	OOM
GNN+Pairwise Info	SEAL	63.37 ± 0.69	95.65 ± 0.29	25.25 ± 3.90	97.97 ± 0.19	48.80 ± 5.61	99.79 ± 0.02	86.93 ± 0.43
	BUDDY	**64.59 ± 0.46**	96.52 ± 0.40	29.60 ± 4.75	99.81 ± 0.02	47.33 ± 1.96	99.56 ± 0.02	**87.86 ± 0.18**
	Neo-GNN	**66.13 ± 0.61**	**98.23 ± 0.05**	20.95 ± 6.03	98.06 ± 2.00	48.45 ± 1.01	97.30 ± 0.14	83.54 ± 0.32
	NCN	63.86 ± 0.51	97.83 ± 0.04	**76.52 ± 10.47**	**99.97 ± 0.00**	**62.63 ± 1.15**	**99.95 ± 0.01**	**89.27 ± 0.05**
	NCNC	**65.97 ± 1.03**	**98.20 ± 0.05**	**70.23 ± 12.11**	**99.97 ± 0.01**	**62.61 ± 0.76**	**99.97 ± 0.01**	**89.82 ± 0.43**
	NBFNet	OOM	OOM	>24h	>24h	OOM	OOM	OOM
	PEG	49.02 ± 2.99	94.45 ± 0.89	30.28 ± 4.92	99.45 ± 0.04	OOM	OOM	OOM

for ogbl-ppa, and MRR for ogbl-citation2). We use “>24h" to denote methods that require more than 24 hours for either training one epoch or evaluation. OOM indicates that the algorithm requires over 50Gb of GPU memory. Since ogbl-ddi has no node features, we mark the MLP results with a “N/A". Additional results in terms of other metrics are presented in Appendix A.6. We have several noteworthy observations concerning the methods, the datasets, the evaluation settings, and the overall results. We highlight the main observations below.

Observation 1: Better than Reported Performance. We find that for some models we are able to achieve superior performance compared to what is reported by recent studies. For instance, in our study Neo-GNN [143] achieves the best overall test performance on ogbl-collab with a Hits@50 of 66.13. In contrast, the reported performance in [143] is only 57.52, which would rank seventh under our current setting. This is because the original study [143] does not follow the standard setting of including validation edges in the graph during testing. This setting, as noted in Section 2.2.3, is used by all other methods on ogbl-collab. However it was omitted by [143], resulting in lower reported performance. Furthermore, on ogbl-citation2 [39], our results for the heuristic methods are typically around 75% MRR. This significantly outperforms previously reported results, which report an MRR of around 50% [146, 15]. The disparity arises as previous studies treat the ogbl-citation2 as a

Table 2.3 Comparison of ours and the reported results for GCN and GAE. Most of results are under-reported.

GCN	ogbl-collab Hits@50	ogbl-ppa Hits@100	ogbl-ddi Hits@20	ogbl-citation2 MRR	GAE	Cora AUC	Citeseer AUC	Pubmed AUC
Reported	47.14 ± 1.45	18.67 ± 1.32	37.07 ± 5.07	84.74 ± 0.21	Reported	91.00 ± 0.01	89.5 ± 0.05	96.4 ± 0.00
Ours	**54.96 ± 3.18**	**29.57 ± 2.90**	49.90 ± 7.23	**84.85 ± 0.07**	Ours	**95.08 ± 0.33**	**97.06 ± 0.22**	**97.47 ± 0.08**

directed graph when applying heuristic methods. However, for GNN-based methods, ogbl-citation2 is typically converted to a undirected graph. We remedy this by also converting ogbl-citation2 to an undirected graph when computing the heuristics, leading to a large increase in performance.

Furthermore, with proper tuning, conventional baselines like GCN [53] and GAE [52] generally exhibit enhanced performance relative to what was originally reported across all datasets. For example, we find that GAE can achieve the second best MRR on Citeseer and GCN the third best Hits@20 on ogbl-ddi. A comparison of the reported results and ours are shown in Table 2.3. We note that we report AUC for Cora, Citeseer, Pubmed as it was used in the original study. These observations suggest that the performance of various methods are better than what was reported in their initial publications. However, many studies [15, 117, 146] only report the original performance for comparison, which has the potential to lead to inaccurate conclusions.

Observation 2: Divergence from Reported Results on ogbl-ddi. We observe that our results in Table 2.2 for ogbl-ddi differ from the reported results. Outside of GCN, which reports better performance, most other GNN-based methods report a lower-than-reported performance. For example, for BUDDY we only achieve a Hits@20 of 29.60 vs. the reported 78.51 (see Appendix A.4 for a comprehensive comparison among methods). We find that the reason for this difference depends on the method. BUDDY [15] reported [1] using 6 negatives per positive sample during training, leading to an increase in performance. Neo-GNN [143] first pretrains the GNN under the link prediction task, and then uses the pretrained model as the initialization for Neo-GNN.[2] For a fair comparison among methods, we only use 1 negative per positive sample in training and we don't apply the pretraining. For other methods, we find that a weak relationship between the validation and test performance complicates the tuning process, making it difficult to find the optimal

[1]https://github.com/melifluos/subgraph-sketching
[2]https://github.com/seongjunyun/Neo-GNNs

hyperparameters. Please see Appendix A.5 for a more in-depth study and discussion.

Observation 3: High Model Standard Deviation. The results in Tables 2.1 and 2.2 present the mean performance and standard deviation when training over 10 seeds. Generally, we find that for multiple datasets the standard deviation of the ranking metrics is often high for most models. For example, the standard deviation for MRR can be as high as 8.82, 8.96, or 7.75 for Cora, Citeseer, and Pubmed, respectively. Furthermore, on ogbl-ddi the standard deviation of Hits@20 reaches as high as 10.47 and 15.56. A high variance indicates unstable model performance. This makes it difficult to compare results between methods as the true performance lies in a larger range. This further complicates replicating model performance, as even large differences with the reported results may still fall within variance (see observation 2). Later in Section 2.4.3 we find that our new evaluation can reduce the model variance for all datasets (see Table 2.6). This suggests that the high variance is related to the current evaluation procedure.

Observation 4: Inconsistency of AUC vs. Ranking-Based Metrics. The AUC score is widely adopted to evaluate recent advanced link prediction methods [52, 157]. However, from our results in Tables 2.1 and 2.2 we observe that there exists a disparity between AUC and ranking-based metrics. In some cases, the AUC score can be high when the ranking metric is very low or even 0. For example, the Shortest Path heuristic records a Hits@K of 0 on ogbl-ppa. However, the AUC score is > 99%. Furthermore, even though RA records the third and fifth best performance on ogbl-ppa and ogbl-collab, respectively, it has a lower AUC score than Shortest Path on both. Previous works [42, 133] argued that AUC is not a proper metric for link prediction. This is due to the inapplicability of AUC for highly imbalanced problems [18, 94].

2.4 New Evaluation Setting

In this section, we introduce a new setting for evaluating link prediction methods. We first discuss the unrealistic nature of the current evaluation setting in Section 2.4.1. Based on this, we present our new evaluation setting in Section 2.4.2, which aims to align better with real-world scenarios. Lastly, in Section 2.4.3, we present and discuss the results based on our new evaluation setting.

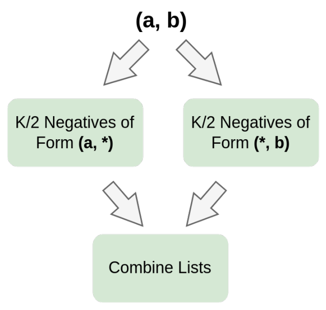
2.4.1 Issues with the Existing Evaluation Setting

The existing evaluation procedure for link prediction is to rank a positive sample against a set of K randomly selected negative samples. The same set of K negatives are used for all positive samples (with the exception of ogbl-citation2 which uses 1000 per positive sample). We demonstrate that there are multiple issues with this setting, making it difficult to properly evaluate the effectiveness of current models.

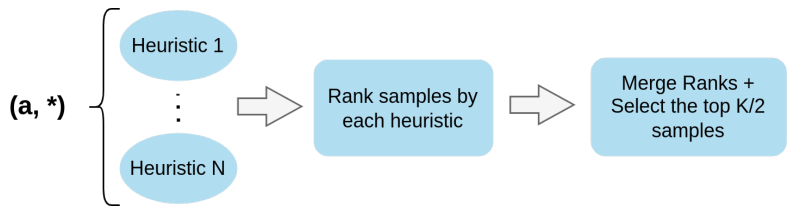
<u>Issue 1</u>: Non-Personalized Negative Samples. The existing evaluation setting uses the same set of negative samples for all positive samples (outside of ogbl-citation2). This strategy, referred to as global negative sampling [118], is not a commonly sought objective. Rather, we are often more interested in predicting links that will occur for a specific node. Take, for example, a social network that connects users who are friends. In this scenario, we may be interested in recommending new friends to a user u. This requires learning a classifier f that assigns a probability to a link existing. When evaluating this task, we want to rank links where u connects to an existing friend above those where they don't. For example, if u is friends with a but not b, we hope that $f(u, a) > f(u, b)$. However, the existing evaluation setting doesn't explicitly test for this. Rather it compares a true sample (u, a) with a potentially unrelated negative sample, e.g., (c, d). This is not aligned with the real-world usage of link prediction on such graphs.

<u>Issue 2</u>: Easy Negative Samples. The existing evaluation setting randomly selects negative samples to use. However given the large size of most graphs (see Table A.1 in Appendix A.3), randomly sampled negatives are likely to choose two nodes that bear no relationship to each other. Such node pairs are trivial to classify. We demonstrate this by plotting the distribution of common neighbors (CN), a strong heuristic, for all positive and negative test samples in Figure 2.1. Almost all the negative samples contain no CNs, making them easy to classify. We further show that the same problem afflicts even the smaller datasets in Figure A.1 in Appendix A.1.

These observations suggest that a more realistic evaluation strategy is desired. At the core of this challenge is which negative samples to use during evaluation. We discuss our design for solving this in the next subsection.



(a) Negative sample generation for one positive sample.


(a, *)
Heuristic 1
Heuristic N
Rank samples by each heuristic
Merge Ranks + Select the top K/2 samples

(b) Process of determining negative samples that contain a node a.

Figure 2.2 Pipeline for generating the hard negative samples for a positive sample (a, b). We employ multiple heuristics to obtain the hard negative samples.

2.4.2 Heuristic Related Sampling Technique (HeaRT)

In this subsection, we introduce new strategy for evaluating link prediction methods. To address the concerns outlined in Section 2.4.1, we design a new method for sampling negatives during evaluation. Our strategy, HeaRT, solves these challenges by: (a) personalizing the negatives to each sample and (b) using heuristics to select hard negative samples. This allows for the negative samples to be directly related to each positive sample while also being non-trivial. We further discuss how to ensure that the negative samples are both personalized and non-trivial for a specific positive sample.

From our discussion in Section 2.4.1, we are motivated in personalizing the negatives to each positive sample. Since the positive samples in the current datasets are node pairs, we seek to personalize the negatives to both nodes in the positive sample. Extending our example in Section 2.4.1, this is analogous to restricting the negatives to contain one of the two users from the original friendship pair. As such, for a positive sample (u, a), the negative samples will belong to the set:

$$S(u, a) = \{(u', a) \mid u' \in \mathcal{V}\} \cup \{(u, a') \mid a' \in \mathcal{V}\}, \tag{2.3}$$

where $\mathcal{V}$ is the set of nodes. This is similar to the setting used for knowledge graph completion (KGC) [7] which uses all such samples for evaluation. However, one drawback of evaluating each positive sample against the entire set of possible corruptions is the high computational cost. To mitigate this issue we consider only utilizing a small subset of $S(u, a)$ during evaluation.

The key challenge is how to generate a subset of $S(u, a)$. If we randomly sample from $S(u, a)$,

Table 2.4 Results on Cora, Citeseer, and Pubmed (%) under HeaRT. Highlighted are the results ranked first (green), second (blue), and third (orange).

	Models	Cora		Citeseer		Pubmed	
		MRR	Hits@10	MRR	Hits@10	MRR	Hits@10
Heuristic	CN	9.78	20.11	8.42	18.68	2.28	4.78
	AA	11.91	24.10	10.82	22.20	2.63	5.51
	RA	11.81	24.48	10.84	22.86	2.47	4.9
	Shortest Path	5.04	15.37	5.83	16.26	0.86	0.38
	Katz	11.41	22.77	11.19	24.84	3.01	5.98
Embedding	Node2Vec	14.47 ± 0.60	32.77 ± 1.29	21.17 ± 1.01	45.82 ± 2.01	3.94 ± 0.24	8.51 ± 0.77
	MF	6.20 ± 1.42	15.26 ± 3.39	7.80 ± 0.79	16.72 ± 1.99	4.46 ± 0.32	9.42 ± 0.87
	MLP	13.52 ± 0.65	31.01 ± 1.71	22.62 ± 0.55	48.02 ± 1.79	6.41 ± 0.25	15.04 ± 0.67
GNN	GCN	**16.61 ± 0.30**	**36.26 ± 1.14**	21.09 ± 0.88	47.23 ± 1.88	7.13 ± 0.27	15.22 ± 0.57
	GAT	13.84 ± 0.68	32.89 ± 1.27	19.58 ± 0.84	45.30 ± 1.3	4.95 ± 0.14	9.99 ± 0.64
	SAGE	14.74 ± 0.69	34.65 ± 1.47	21.09 ± 1.15	48.75 ± 1.85	**9.40 ± 0.70**	**20.54 ± 1.40**
	GAE	**18.32 ± 0.41**	**37.95 ± 1.24**	**25.25 ± 0.82**	**49.65 ± 1.48**	5.27 ± 0.25	10.50 ± 0.46
GNN+Pairwise Info	SEAL	10.67 ± 3.46	24.27 ± 6.74	13.16 ± 1.66	27.37 ± 3.20	5.88 ± 0.53	12.47 ± 1.23
	BUDDY	13.71 ± 0.59	30.40 ± 1.18	22.84 ± 0.36	48.35 ± 1.18	7.56 ± 0.18	16.78 ± 0.53
	Neo-GNN	13.95 ± 0.39	31.27 ± 0.72	17.34 ± 0.84	41.74 ± 1.18	**7.74 ± 0.30**	**17.88 ± 0.71**
	NCN	14.66 ± 0.95	35.14 ± 1.04	**28.65 ± 1.21**	**53.41 ± 1.46**	5.84 ± 0.22	13.22 ± 0.56
	NCNC	14.98 ± 1.00	**36.70 ± 1.57**	**24.10 ± 0.65**	**53.72 ± 0.97**	**8.58 ± 0.59**	**18.81 ± 1.16**
	NBFNet	13.56 ± 0.58	31.12 ± 0.75	14.29 ± 0.80	31.39 ± 1.34	>24h	>24h
	PEG	**15.73 ± 0.39**	36.03 ± 0.75	21.01 ± 0.77	45.56 ± 1.38	4.4 ± 0.41	8.70 ± 1.26

we risk only utilizing easy negative samples. This is one of the issues of the existing evaluation setting (see Issue 2 in Section 2.4.1), whereby randomly selecting negatives, they unknowingly produce negative samples that are too easy. We address this by selecting the negative samples via a combination of multiple heuristics. Since heuristics typically correlate well with performance, we ensure that the negative samples will be non-trivial to classify. This is similar to the concept of candidate generation [32, 26], which only ranks a subset of candidates that are most likely to be true.

An overview of the generation process is given in Figure 2.2. For each positive sample, we generate K negative samples. To allow personalization to both nodes in the positive sample equally, we sample $K/2$ negatives with each node. For the heuristics, we consider RA [155], PPR [9], and feature similarity. A more detailed discussion on the negative sample generation is given in Appendix A.7. It's important to note that our work centers specifically on negative sampling during the evaluation stage (validation and test). This is distinct from prior work that concerns the negatives sampled used during the training phase [134, 91]. As such, the training process remains unaffected under both the existing and HeaRT setting.

2.4.3 Results and Discussion

In this subsection we present our results when utilizing HeaRT. We follow the parameter ranges introduced in Section 2.2.3. For all datasets we use $K = 500$ negative samples per positive sample during evaluation. Furthermore for ogbl-ppa we only use a small subset of the validation and test positive samples (100K each) for evaluation. This is because the large size of the validation and test sets (see Table A.1 in Appendix A.3) makes HeaRT prohibitively expensive.

The results are shown in Table 2.4 (Cora, Citeseer, Pubmed) and Table 2.5 (OGB). For simplicity, we only include the MRR and Hits@10 for Cora, Citeseer, Pubmed, and the MRR and Hits@20 for OGB. Additional results for other metrics can be found in Appendix A.9. We note that most datasets, outside of ogbl-ppa, exhibit much lower performance than under the existing setting. This is though we typically use much fewer negative samples in the new setting, implying that the negative samples produced by HeaRT are much harder. We highlight the main observations below.

Observation 1: Better Performance of Simple Models. We find that under HeaRT, “simple" baseline models (i.e., heuristic, embedding, and GNN methods) show a greater propensity to outperform their counterparts via ranking metrics than under the existing setting. Specifically, we focus on MRR in Table 2.1, 2.4, and 2.5, and the corresponding ranking-based metrics in Table 2.2. Under the existing setting, such methods only rank in the top three for any dataset a total of 5 times. However, under HeaRT this occurs 10 times. Furthermore, under the existing setting only 1 “simple" method ranks best overall while under HeaRT there are 4. This suggests that recent advanced methods may have benefited from the easier negative samples in the existing setting.

Another interesting observation is that on ogbl-collab, heuristic methods are able to outperform more complicated models by a large margin. Specifically, we find that Katz is the best ranked method, Shortest Path the second, and RA the fourth. Furthermore, the MRR gap between the second ranked method (Shortest Path) and the third (BUDDY) is very large at 14.29 points. We observe that this result is caused by the dynamic nature of the graph, where node pairs that are linked in the training data may also be present as positive samples in the test. We further expound on this observation in Appendix A.8.

Table 2.5 Results on OGB datasets (%) under HeaRT. Highlighted are the results ranked first (green), second (blue), and third (orange).

Models	ogbl-collab		ogbl-ddi		ogbl-ppa		ogbl-citation2	
	MRR	Hits@20	MRR	Hits@20	MRR	Hits@20	MRR	Hits@20
CN	12.60	27.51	6.71	38.69	25.70	68.25	17.11	41.73
AA	16.40	32.65	6.97	39.75	26.85	70.22	17.83	43.12
RA	28.14	41.16	8.70	44.01	28.34	71.50	17.79	43.34
Shortest Path	**46.71**	**46.56**	0	0	0.54	1.31	>24h	>24h
Katz	**47.15**	**48.66**	6.71	38.69	25.70	68.25	14.10	35.55
Node2Vec	12.10 ± 0.20	25.85 ± 0.21	11.14 ± 0.95	63.63 ± 2.05	18.33 ± 0.10	53.42 ± 0.11	14.67 ± 0.18	42.68 ± 0.20
MF	26.86 ± 1.74	38.44 ± 0.07	**13.99 ± 0.47**	59.50 ± 1.68	22.47 ± 1.53	70.71 ± 4.82	8.72 ± 2.60	29.64 ± 7.30
MLP	12.61 ± 0.66	23.05 ± 0.89	N/A	N/A	0.98 ± 0.00	1.47 ± 0.00	16.32 ± 0.07	43.15 ± 0.10
GCN	18.28 ± 0.84	32.90 ± 0.66	**13.46 ± 0.34**	64.76 ± 1.45	26.94 ± 0.48	68.38 ± 0.73	19.98 ± 0.35	**51.72 ± 0.46**
GAT	10.97 ± 1.16	29.58 ± 2.42	**12.92 ± 0.39**	**66.83 ± 2.23**	OOM	OOM	OOM	OOM
SAGE	20.89 ± 1.06	33.83 ± 0.93	12.60 ± 0.72	**67.19 ± 1.18**	27.27 ± 0.30	69.49 ± 0.43	**22.05 ± 0.12**	**53.13 ± 0.15**
GAE	OOM	OOM	3.49 ± 1.73	17.81 ± 9.80	OOM	OOM	OOM	OOM
SEAL	22.53 ± 3.51	36.48 ± 2.55	9.99 ± 0.90	49.74 ± 2.39	**29.71 ± 0.71**	**76.77 ± 0.94**	**20.60 ± 1.28**	48.62 ± 1.93
BUDDY	**32.42 ± 1.88**	**45.62 ± 0.52**	12.43 ± 0.50	58.71 ± 1.63	27.70 ± 0.33	71.50 ± 0.68	19.17 ± 0.20	47.81 ± 0.37
Neo-GNN	21.90 ± 0.65	38.40 ± 0.29	10.86 ± 2.16	51.94 ± 10.33	21.68 ± 1.14	64.81 ± 2.26	16.12 ± 0.25	43.17 ± 0.53
NCN	17.51 ± 2.50	37.07 ± 2.97	12.86 ± 0.78	**65.82 ± 2.66**	**35.06 ± 0.26**	**81.89 ± 0.31**	**23.35 ± 0.28**	**53.76 ± 0.20**
NCNC	19.02 ± 5.32	35.67 ± 6.78	>24h	>24h	**33.52 ± 0.26**	**82.24 ± 0.40**	19.61 ± 0.54	51.69 ± 1.48
NBFNet	OOM	OOM	>24h	>24h	OOM	OOM	OOM	OOM
PEG	15.68 ± 1.10	29.74 ± 0.95	12.05 ± 1.14	50.12 ± 6.55	OOM	OOM	OOM	OOM

<u>Observation 2</u>: Lower Model Standard Deviation. We observed earlier that, under the existing evaluation setting, the model variance across seeds was high (see observation 3 in Section 2.3). This complicates model comparison as the model performance is unreliable. Interestingly, we find that HeaRT is able to dramatically reduce the variance for all datasets. We demonstrate this by first calculating the mean standard deviation across all models on each individual dataset. This was done for both evaluation settings with the results compared. As demonstrated in Table 2.6, the mean standard deviation decreases for all datasets. This is especially true for Cora, Citeseer, and Pubmed, which each decrease by over 85%. Such a large decrease in standard deviation is noteworthy as it allows for a more trustworthy and reliable comparison between methods.

We posit that this observation is caused by a stronger alignment between the positive and negative samples under our new evaluation setting. Under the existing evaluation setting, the same set of negative samples is used for all positive samples. One consequence of this is that a single positive sample may bear little to no relationship to the negative samples (see Section 2.4.1 for more discussion). However, under our new evaluation setting, the negatives for a positive sample are a subset of the corruptions of that sample. This allows for a more natural comparison via

Table 2.6 Mean model standard deviation for the existing setting and HeaRT. We use Hits@20 for ogbl-ddi, Hits@50 for ogbl-collab, Hits@100 for ogbl-ppa, and MRR otherwise.

Dataset	Existing	HeaRT	% Change
Cora	5.19	0.79	-85%
Citeseer	5.94	0.88	-85%
Pubmed	4.14	0.35	-92%
ogbl-collab	1.49	0.96	-36%
ogbl-ppa	2.13	0.36	-83%
ogbl-ddi	6.77	3.49	-48%
ogbl-citation2	1.39	0.59	-58%

ranking-based metrics as the samples are more related and can be more easily compared.

Observation 3: Lower Model Performance. We observe that the majority of datasets exhibit a significantly reduced performance in comparison to the existing setting. For example, under the existing setting, models typically achieve a MRR of around 30, 50, and 30 on Cora, Citeseer, and Pubmed (Table 2.1), respectively. However, under HeaRT the MRR for those datasets is typically around 20, 25, and 10 (Table 2.4). Furthermore for ogbl-citation2, the MRR of the best performing model falls from a shade under 90 on the existing setting to slightly over 20 on HeaRT. Lastly, we note that the performance on ogbl-ppa actually increases. This is because we only utilize a small subset of the total test set when evaluating on HeaRT, nullifying any comparison between the two settings.

These outcomes are observed despite HeaRT using much fewer negative samples than the original setting. This suggests that the negative samples generated by HeaRT are substantially more challenging than those used in the existing setting. This underscores the need to develop more advanced methodologies that can tackle harder negatives samples like in HeaRT.

2.5 Conclusion

In this work we have revealed several pitfalls found in recent works on link prediction. To overcome these pitfalls, we first establish a benchmarking that facilitates a fair and consistent evaluation across a diverse set of models and datasets. By doing so, we are able to make several illuminating observations about the performance and characteristics of various models. Furthermore, based on several limitations we observed in the existing evaluation procedure, we introduce a more practical setting called HeaRT (Heuristic Related Sampling Technique). HeaRT incorporates a more real-world evaluation setting, resulting in a better comparison among methods. By introducing a more rigorous and realistic assessment, HeaRT could guide the field towards more effective models, thereby advancing the state of the art in link prediction.

CHAPTER 3

EVALUATION PITFALLS IN KNOWLEDGE GRAPH

Knowledge graphs (KGs) facilitate a wide variety of applications. Despite great efforts in creation and maintenance, even the largest KGs are far from complete. Hence, KG completion (KGC) has become one of the most crucial tasks for KG research. Recently, considerable literature in this space has centered around the use of Message Passing (Graph) Neural Networks (MPNNs), to learn powerful embeddings. The success of these methods is naturally attributed to the use of MPNNs over simpler multi-layer perceptron (MLP) models, given their additional message passing (MP) component. In this work, we find that surprisingly, simple MLP models are able to achieve comparable performance to MPNNs, suggesting that MP may not be as crucial as previously believed. With further exploration, we show careful scoring function and loss function design has a much stronger influence on KGC model performance. This suggests a conflation of scoring function design, loss function design, and MP in prior work, with promising insights regarding the scalability of state-of-the-art KGC methods today, as well as careful attention to more suitable MP designs for KGC tasks tomorrow.

3.1 Introduction

Knowledge graphs (KGs) [5, 14] are a type of knowledge base, which store multi-relational factual knowledge in the form of triplets. Each triplet specifies the relation between a head and a tail entity. KGs conveniently capture rich structured knowledge about many types of entities (e.g. objects, events, concepts) and thus facilitate numerous applications such as information retrieval [129], recommender systems [116], and question answering [121]. To this end, the adopted KGs are expected to be as comprehensive as possible to provide all kinds of required knowledge. However, existing large-scale KGs are known to be far from complete with large portions of triplets missing [5, 14]. Imputing these missing triplets is of great importance. Furthermore, new knowledge (triplets) is constantly emerging even between existing entities, which also calls for dedicated efforts to predict these new triplets [30, 47]. Therefore, *knowledge graph completion* (KGC) is a problem of paramount importance [67, 140]. A crucial step towards better KGC performance is to learn low-dimensional continuous embeddings for both entities and relations [8].

Recently, due to the intrinsic graph-structure of KGs, Graph Neural Networks (GNNs) have been adopted to learning more powerful embeddings for their entities and relations, and thus facilitate the KGC. There are mainly two types of GNN-based KGC methods: Message Passing Neural Networks (MPNNs) [96, 110] and path-based methods [157, 149, 156]. In this work, we focus on MPNN-based models, which update node features through a message passing (MP) process over the graph where each node collects and transforms features from its neighbors. When adopting MPNNs for KGs, dedicated efforts are often devoted to developing more sophisticated MP processes that are customized for better capturing multi-relational information [110, 96, 138]. The improvement brought by MPNN-based models is thus naturally attributed to these enhanced MP processes. Therefore, current research on developing better MPNNs for KGs is still largely focused on advancing MP processes.

Present Work. In this work, we find that, surprisingly, the MP in the MPNN-based models is not the most critical reason for reported performance improvements for KGC. Specifically, we replaced MP in several state-of-the-art KGC-focused MPNN models such as RGCN [96], CompGCN [110] and KBGAT [79] with simple Multiple Layer Perceptrons (MLPs) and achieved comparable performance to their corresponding MPNN-based models, across a variety of datasets and implementations. We carefully scrutinized these MPNN-based models and discovered they also differ from each other in other key components such as scoring functions and loss functions. To better study how these components contribute to the model, we conducted comprehensive experiments to demonstrate the effectiveness of each component. Our results indicate that the scoring and loss functions have stronger influence while MP makes almost no contributions. Based on our findings, we develop ensemble models built upon MLPs, which are able to achieve better performance than MPNN-based models; these implications are powerful in practice, given scalability advantages of MLPs over MPNNs [147].

3.2 Related Work

There are mainly two types of GNN-based KGC models: MPNN-based models and path-based models. When adopting MPNNs for KG, recent efforts have been made to deal with the multi-relational edges in KGs by designing MP operations. RGCN [96] introduces the relation-specific transformation matrices. CompGCN [110] integrates neighboring information based on entity-relation composition operations. KBGAT [79] learns attention coefficients to distinguish the role of entity in various relations. Path-based models learn pair-wise representations by aggregating the path information between the two nodes. NBFNet [157] integrates the information from all paths between the two nodes. RED-GNN [149] makes use of the dynamic programming and A*Net [156] prunes paths by prioritizing important nodes and edges. In this paper, we focus on investigating how the MP component in the MPNNs affects their performance in the KGC task. Hence, we do not include these path-based models into the comparison. A concurrent work [150] has similar observations as ours. However, they majorly focus on exploring how the MP component affects the performance. Our work provides a more thorough analysis on the major contributors for MPNN-based KGC models and proposes a strong ensemble model based upon the analysis.

3.3 Preliminaries

Before moving to main content, we first introduce KGC-related preliminaries, five datasets and three MPNN-based models we adopt for investigations.

3.3.1 Knowledge graph completion (KGC)

The task of KGC is to infer missing triplets based on known facts in the KG. In KGC, we aim to predict a missing head or tail entity given a triplet. Specifically, we denote the triplets with missing head (tail) entity as $(h, r, ?)$ $((?, r, t))$, where the question mark indicates the entity we aim to predict. Since the head entity prediction and tail entity prediction tasks are symmetric, in the following, we only use the tail entity prediction task for illustration. When conducting the KGC task for a triplet $(h, r, ?)$, we use all entities in KG as candidates and try to select the best one as the tail entity. Typically, for each candidate entity t', we evaluate its score for the triplet (h, r, t') with the function $s_{h,r}(t') = f(h, r, t')$, where $s_{h,r}(t')$ is the score of t' given the head entity h and the

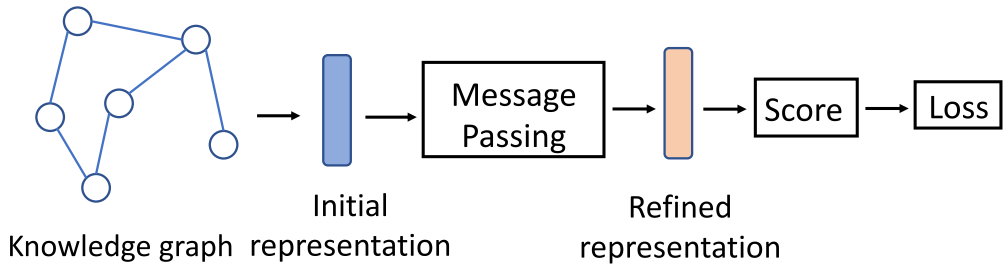


Figure 3.1 A general MPNN framework for KGC. It has three key components: message passing, scoring function and loss function.

relation r, and f is a scoring function. We choose the entity t' with the largest score as the predicted tail entity. $f(\cdot)$ can be modeled in various ways as discussed later.

Datasets. We use five well-known KG datasets, i.e., **FB15k** [8], **FB15k-237** [109, 108], **WN18** [96], **WN18RR** [21] and **NELL-995** [130] for this study. The detailed descriptions and data statistics can be found in Appendix B.1. Following the settings in previous works [110, 96], triplets in these datasets are randomly split into training, validation, and test sets, denoted $\mathcal{D}_{train}, \mathcal{D}_{val}, \mathcal{D}_{test}$, respectively. The triplets in the training set are regarded as the known facts. We manually remove the head/tail entity of the triplets in the validation and test sets for model selection and evaluation. Specifically, for the tail entity prediction task, given a triplet (h, r, t^*), we remove t^* and construct a test sample $(h, r, ?)$. The tail entity t^* is regarded as the ground-truth for this sample.

Evaluation Metrics. When evaluating the performance, we focus on the predicted scores for the ground-truth entity of the triplets in the test set $\mathcal{D}_{test}$. For each triplet $(h, r, ?)$ in the test set, we sort all candidate entities t in a non-increasing order according to $s_{h,r}(t)$. Then, we use the rank-based measures to evaluate the prediction quality, including Mean Reciprocal Rank (**MRR**) and **Hits@N**. In this work, we choose $N \in \{1, 3, 10\}$. See Appendix B.2 for their definitions.

3.3.2 MPNN-based KGC

Various MPNN-based models have been utilized for KGC by learning representations for the entities and relations of KGs. The learnt representations are then used as input to a scoring function $f(\cdot)$. Next, we first introduce MPNN models specifically designed for KG. Then, we introduce scoring functions. Finally, we describe the training process, including loss functions.

3.3.2.1 MPNNs for learning KG representations

KGs can be naturally treated as graphs with triplets being the relational edges. When MPNN models are adapted to learn representations for KGs, the MP process in the MPNN layers is tailored for handling such relational data (triplets). In this paper, we investigate three representative MPNN-based models for KGC, i.e., **CompGCN** [110], **RGCN** [96] and **KBGAT** [79], which are most widely adopted. As in standard MPNN models, these models stack multiple layers to iteratively aggregate information throughout the KG. Each intermediate layer takes the output from the previous layer as the input, and the final output from the last layer serves as the learned embeddings. In addition to entity embeddings, some MPNN-based models also learn relation embeddings. For a triplet (h, r, t), we use $\mathbf{x}_h^{(k)}$, $\mathbf{x}_r^{(k)}$, and $\mathbf{x}_t^{(k)}$ to denote the head, relation, and tail embeddings obtained after the k-th layer. Specifically, the input embeddings of the first layer $\mathbf{x}_h^{(0)}$, $\mathbf{x}_r^{(0)}$ and $\mathbf{x}_t^{(0)}$ are randomly initialized. RGCN aggregates neighborhood information with the relation-specific transformation matrices. CompGCN defines direction-based transformation matrices and introduces relation embeddings to aggregate the neighborhood information. It introduces the composition operator to combine the embeddings to leverage the entity-relation information. KBGAT proposes attention-based aggregation process by considering both the entity embedding and relation embedding. More details about the MP process for CompGCN, RGCN and KBGAT can be found in Appendix B.3. For MPNN-based models with K layers, we use $\mathbf{x}_h^{(K)}$, $\mathbf{x}_r^{(K)}$, and $\mathbf{x}_t^{(K)}$ as the final output embeddings and denote them as $\mathbf{x}_h$, $\mathbf{x}_r$, and $\mathbf{x}_t$ for the simplicity of notations. Note that RGCN does not involve relation embedding $\mathbf{x}_r$ in the MP process, which will be randomly initialized if required by the scoring function.

3.3.2.2 Scoring functions

After obtaining the final embeddings from the MP layers, they are utilized as input to the scoring function f. Various scoring functions can be adopted. Two widely used scoring functions are DistMult [131] and ConvE [21]. More specifically, RGCN adopts DistMult. In CompGCN, both scoring functions are investigated and ConvE is shown to be more suitable in most cases. Hence, in this paper, we use ConvE as the default scoring function for CompGCN. See Appendix B.4 for

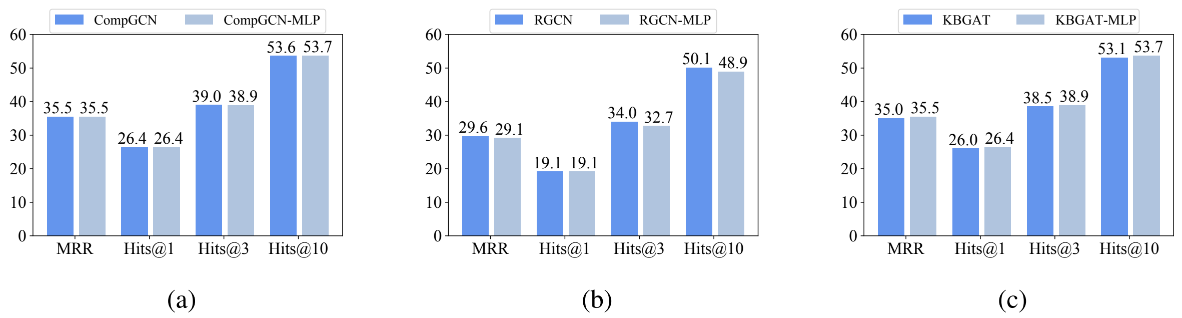


Figure 3.2 KGC results (%) of CompGCN/CompGCN-MLP, RGCN/RGCN-MLP, and KBGAT/KBGAT-MLP on `FB15K-237`. The MLP counterparts achieve compare performance as the corresponding MPNN models.

more scoring function details.

3.3.2.3 Training MPNN-based models for KGC

To train the MPNN model, the KGC task is often regarded as a binary classification task to differentiate the true triplets from the randomly generated “fake” triplets. During training, all triplets in $\mathcal{D}_{train}$ and the corresponding inverse triplets $\mathcal{D}'_{train} = \{(t, r_{in}, h)|(h, r, t) \in \mathcal{D}_{train}\}$ are treated as positive samples, where r_{in} is the inverse relation of r. The final positive sample set can be denoted as $\mathcal{D}^*_{train} = \mathcal{D}_{train} \bigcup \mathcal{D}'_{train}$. Negative samples are generated by corrupting the triplets in $\mathcal{D}^*_{train}$. Specifically, for a triplet $(e_1, rel, e_2) \in \mathcal{D}^*_{train}$, we corrupt it by replacing its tail entities with other entities in the KG. More formally, the set of negative samples corresponding to the triplet (e_1, rel, e_2) is denoted as: $C_{(e_1,rel,e_2)} = \{(e_1, rel, e'_2)|e'_2 \in \mathcal{V}, e'_2 \neq e_2\}$ where $\mathcal{V}$ is the set of entities in KG. CompGCN uses $C_{(e_1,rel,e_2)}$ as the negative samples. However, not all negative samples are utilized for training the RGCN model. Instead, for each positive sample triplet in $\mathcal{D}^*_{train}$, they adopt negative sampling to select 10 such samples from $C_{(e_1,rel,e_2)}$, and use only these for training. Also, for RGCN, any relation r and its inverse relation r_{in} share the same diagonal matrix for DistMult in Eq. (B.5) in Appendix B.4. Both CompGCN and RGCN adopt the Binary Cross-Entropy (BCE) loss. More details are given in Appendix B.5.

3.3.2.4 Major differences between MPNNs

We demonstrate an overview of MPNN-based model frameworks for the KGC task in Figure 5.1. Specifically, the framework consists of several key components including the MP (introduced in

Section 3.3.2.1), the scoring function (Section 3.3.2.2), and the loss function (Section 3.3.2.3). Training can typically be conducted end-to-end. Both RGCN and CompGCN follow this framework with various designs in each component. We provide a more detailed comparison about them later in this section. However, KBGAT adopts a two-stage training process, which separates the training of the MP process (representation learning) and the scoring function. KBGAT achieves strong performance as reported in the original paper [79], which was later attributed to a test leakage issue [105]. After addressing this test leakage issue, we found that fitting KBGAT into the general framework described in Figure 5.1 leads to much higher performance than training it with the two-stage process (around 10% improvement on `FB15K-237`). Hence, in this paper, we conduct analyses for KBGAT by fitting its MP process (described in Appendix B.3) into the framework described in Figure 5.1.

We summarize the major differences between RGCN, CompGCN, and KBGAT across three major components: **(1) Message Passing.** Their MP processes are different as described in Section 3.3.2.1 and detailed in Appendix B.3. **(2) Scoring Function.** They adopt different scoring functions. RGCN adopts the DistMult scoring function while CompGCN achieves best performance with ConvE. Thus, in this paper, we use ConvE as its default scoring function. For KBGAT, we adopt ConvE as its default scoring function. **(3) Loss Function.** As described in Section 3.3.2.3, CompGCN utilizes all entities in the KG as negative samples for training, while RGCN adopts a negative sampling strategy. For KBGAT, we also utilize all entities to construct negative samples, similar to CompGCN.

3.4 What Matters for MPNN-based KGC?

Recent efforts in adapting MPNN models for KG mostly focus on designing more sophisticated MP components to better handle multi-relational edges. These recently proposed MPNN-based methods have reported strong performance on the KGC task. Meanwhile, RGCN, CompGCN and KBGAT achieve different performance. Their strong performance compared to traditional embedding based models and their performance difference are widely attributed to the MP components [96, 110, 79]. However, as summarized in Section 3.3.2.4, they differ from each other in several ways besides MP;

little attention has been paid to understand how each component affects these models. Thus, what truly matters for MPNN-based KGC performance is still unclear. To answer this question, we design careful experiments to ablate the choices of these components in RGCN, CompGCN and KBGAT to understand their roles, across multiple datasets. All reported results are mean and standard deviation over three seeds. Since MP is often regarded as the major contributor, we first investigate: is MP really helpful? Subsequently, we study the impact of the scoring function and the loss function.

Table 3.1 KGC results (%) with various scoring functions. Models behave differently with different scoring functions.

		`FB15K-237`				`WN18RR`				`NELL-995`			
		MRR	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10
CompGCN	DistMult	33.7± 0.1	24.7±0.1	36.9±0.2	51.5±0.2	42.9±0.1	39.0±0.1	43.9±0.1	51.7±0.3	32.3±0.5	24.3±0.6	36.1±0.4	47.4±0.2
	ConvE	35.5±0.1	26.4±0.1	39.0±0.2	53.6±0.3	47.2±0.2	43.7±0.3	48.5±0.3	54.0±0.0	38.1±0.4	30.4± 0.5	42.2±0.3	52.9± 0.1
RGCN	DistMult	29.6±0.3	19.1±0.5	34.0± 0.2	50.1±0.2	43.0±0.2	38.6±0.3	45.0±0.1	50.8±0.3	27.8±0.2	19.9±0.2	31.4±0.0	43.0±0.3
	ConvE	29.6±0.4	20.3± 0.4	32.7±0.5	47.9±0.6	28.9± 0.7	17.4± 0.8	36.9±0.5	48.8±0.5	31.7±0.2	23.3±0.2	35.3±0.3	48.5± 0.2
KBGAT	DistMult	33.4±0.1	24.5±0.1	36.6±0.1	51.3±0.5	42.1±0.4	38.7±0.4	43.1± 0.6	49.6±0.6	33.0±0.2	25.5± 0.1	36.8± 0.5	47.3±0.5
	ConvE	35.0±0.3	26.0±0.3	38.5± 0.3	53.1± 0.3	46.4±0.2	42.6±0.2	47.9±0.3	53.9±0.2	37.4±0.6	29.7±0.7	41.4± 0.8	52.0±0.4

Table 3.2 KGC results (%) with various loss functions. The loss function significantly impacts model performance.

		`FB15K-237`				`WN18RR`				`NELL-995`			
		MRR	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10
CompGCN	with	31.5±0.1	22.2±0.1	34.8±0.2	49.6±0.2	32.9± 0.9	24.4± 1.5	39.0±0.5	46.7±0.4	32.0±0.2	23.8±0.2	35.7±0.1	48.1±0.2
	w/o	35.5±0.1	26.4±0.1	39.0±0.2	53.6±0.3	47.2±0.2	43.7±0.3	48.5±0.3	54.0±0.0	38.1±0.4	30.4± 0.5	42.2±0.3	52.9± 0.1
RGCN	with	29.6±0.3	19.1±0.5	34.0±0.2	50.1±0.2	43.0±0.2	38.6±0.3	45.0±0.1	50.8±0.3	27.8±0.2	19.9±0.2	31.4±0.0	43.0±0.3
	w/o	33.4 ± 0.1	24.3± 0.1	36.7±0.1	51.4± 0.2	44.5± 0.1	40.9±0.1	45.5± 0.1	51.8± 0.2	34.6± 0.6	27.0± 0.6	38.3± 0.6	49.4± 0.6
KBGAT	with	30.1± 0.3	21.0± 0.3	33.2± 0.4	48.1± 0.3	30.1± 0.2	18.6 ±0.3	37.8±0.3	49.8±0.2	32.6 ± 0.3	24.3± 0.3	36.3± 0.4	48.7± 0.5
	w/o	35.0±0.3	26.0±0.3	38.5± 0.3	53.1± 0.3	46.4±0.2	42.6±0.2	47.9±0.3	53.9±0.2	37.4±0.6	29.7±0.7	41.4± 0.8	52.0±0.4

3.4.1 Does Message Passing Really Help KGC?

For RGCN and CompGCN, we follow the settings in the original papers to reproduce their reported performance. For KBGAT, we follow the same setting of CompGCN as mentioned in Section 3.3.2.4. Specifically, we run these three models on datasets in their original papers. Namely, we run RGCN on `FB15K-237`, `WN18`and `FB15K`, CompGCN on `FB15K-237`and `WN18RR`, and KBGAT on `FB15K-237`, `WN18RR`and `NELL-995`. To understand the role of the MP component, we keep other components untouched and replace their MP components with a simple MLP, which has the same number of layers and hidden dimensions with the corresponding MPNN-based models; note that since an MPNN layer is simply an aggregation over the graph combined with a feature transformation [73], replacing the MP component with MLP can also be achieved by replacing the adjacency matrix of the graph with an identity matrix. We denote the MLP models corresponding to RGCN, CompGCN and KBGAT as RGCN-MLP, CompGCN-MLP and KBGAT-MLP, respectively. We present results for CompGCN, RGCN and KBGAT[1] on the FB15K-237in Figure 3.2. Due to the space limit, we present results on other datasets in Appendix B.6. We summarize the key observation from these figures:

Observation 1 *The counterpart MLP-based models (RGCN-MLP, CompGCN-MLP and KBGAT-MLP) achieve comparable performance to their corresponding MPNN-based models on all datasets, suggesting that MP does not significantly improve model performance.*

To further verify this observation, we investigate how the model performs when the graph structure utilized for MP is replaced by random generated graph structure. We found that the MPNN-based models still achieve comparable performance, which further verifies that the MP is not the major contributor. More details are in Appendix B.7.

Moreover, comparing RGCN with CompGCN on FB15K-237in Figure 3.2, we observe very different performances, while also noting that Observation 1 clarifies that the difference in MP components is not the main contributor. This naturally raises a question: what are the important contributors? According to Section 3.3.2.4, RGCN and CompGCN also adopt different scoring and loss functions, which provides some leads in answering this question. Correspondingly, we next empirically analyze the impact of the scoring and the loss functions with comprehensive experiments. Note that FB15Kand WN18suffer from the inverse relation leakage issue [108, 21]: a large number of test triplets can be obtained from inverting the triplets in the training set. Hence, to prevent these inverse relation leakage from affecting our studies, we conduct experiments on three datasets NELL-995, FB15K-237and WN18RR, where FB15K-237and WN18RR are the filtered versions of FB15K and WN18 after addressing these leakage issues.

[1]We conduct a similar experiment using the setting in the original KBGAT paper. We find that KBGAT and KBGAT-MLP have similar performance on FB15K-237, WN18RRand NELL-995, which is consistent with Observation 1.

Table 3.3 KGC results (%) with varying number of negative samples in the loss function. Generally, utilizing 10 negative samples is not enough. For different datasets and methods, the optimal number of negative samples varies.

		FB15K-237				WN18RR				NELL-995			
	#Neg	MRR	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10
CompGCN	10	31.5±0.1	22.2± 0.1	34.8± 0.2	49.6±0.2	32.9± 0.9	24.4±1.5	39.0± 0.5	46.7± 0.4	32.0± 0.2	23.8± 0.2	35.7± 0.1	48.1±0.2
	50	34.3± 0.1	24.7± 0.1	38.1± 0.1	53.0± 0.1	40.0± 0.4	33.0±0.7	44.0±0.2	51.6± 0.1	37.2± 0.9	28.7±0.9	41.6± 0.9	53.1± 1.0
	200	35.3±0.3	25.5± 0.1	39.2 ± 0.1	53.8± 0.1	43.6± 0.5	39.2±0.7	45.3± 0.2	52.3± 0.5	39.2± 0.3	31.0± 0.3	43.6± 0.2	54.3 ± 0.2
	0.5*N*	34.6 ±0.1	25.3±0.1	38.3± 0.1	52.7±0.1	44.0± 0.5	40.6±0.6	45.1± 0.6	50.9± 0.3	40.7±0.2	33.4± 0.2	44.4± 0.3	54.7±0.2
	N	34.2±0.1	25.0±0.2	37.9± 0.3	52.2± 0.1	44.0±0.3	40.7 ±0.2	45.1± 0.3	50.8±0.5	40.3± 0.5	33.0± 0.5	44.0± 0.5	54.4±0.4
RGCN	10	29.6± 0.3	19.1±0.5	34.0± 0.2	50.1±0.2	43.0±0.2	38.6±0.3	45.0±0.1	50.8±0.3	27.8±0.2	19.9±0.2	31.4±0.0	43.0±0.3
	50	32.5±0.2	22.5±0.3	36.7±0.1	52.0± 0.4	43.9± 0.1	39.6± 0.1	45.6±0.2	51.8± 0.2	29.6±0.3	21.7± 0.3	33.2± 0.3	44.6±0.3
	200	33.2±0.1	23.2± 0.1	37.6±0.2	52.2±0.3	44.1±0.2	39.9± 0.4	45.7±0.2	52.0± 0.2	30.0±0.3	21.7±0.2	34.3±0.4	45.7±0.3
	0.5*N*	33.3±0.2	24.3±0.3	36.9±0.1	50.9±0.2	44.4±0.2	40.7± 0.3	45.5±0.2	52.0± 0.3	33.6±0.3	26.7± 0.3	37.2±0.2	46.7±0.2
	N	33.0±0.4	23.9±0.6	36.5± 0.3	50.5±0.3	44.5±0.2	40.6±0.2	45.8± 0.2	52.4±0.3	33.7± 0.0	26.9± 0.0	37.0± 0.1	46.4± 0.2
KBGAT	10	30.1±0.3	21.0± 0.3	33.2± 0.4	48.1± 0.3	30.1± 0.2	18.6 ±0.3	37.8±0.3	49.8±0.2	32.6 ±0.3	24.3±0.3	36.3±0.4	48.7±0.5
	50	33.6±0.2	24.2±0.3	37.3±0.3	51.9±0.2	35.6±0.5	25.7±0.9	42.6±0.3	51.3±0.1	37.4±0.3	29.0±0.3	41.9±0.2	53.6±0.3
	200	34.7±0.2	25.1±0.2	38.8±0.2	53.3±0.2	39.6±2.3	32.7±3.7	43.4±0.8	51.6± 0.4	39.2±0.2	31.0± 0.3	43.6±0.1	54.3±0.1
	0.5*N*	34.0± 0.1	24.7± 0.1	37.7±0.1	52.2±0.1	44.3±0.1	40.8± 0.3	45.5± 0.2	51.1± 0.3	40.1±0.1	33.2±0.1	43.5±0.2	53.2±0.2
	N	33.6±0.1	24.4±0.2	37.3±0.2	52.0±0.3	43.8± 0.9	40.1± 1.4	45.3± 0.5	51.1± 0.4	39.6± 0.2	32.8± 0.3	43.0±0.3	52.9±0.1

3.4.2 Scoring Function Impact

Next, we investigate the impact of the scoring function on CompGCN, RGCN and KBGAT while fixing their loss function and experimental setting mentioned in Section 3.3.2.4. The KGC results are shown in Table 3.1. In the original setting, CompGCN and KBGAT use ConvE as the scoring function while RGCN adopts DistMult. In Table 3.1, we further present the results of CompGCN and KBGAT with DistMult and RGCN with ConvE. Note that we only make changes to the scoring function, while fixing all the other settings. Hence, in Table 3.1, we still use RGCN, CompGCN and KBGAT to differentiate these three models but use DistMult and ConvE to indicate the specific scoring functions adopted.

From this table, we have several observations: **(1)** In most cases, CompGCN, RGCN and KBGAT behave differently when adopting different scoring functions. For instance, CompGCN and KBGAT achieve better performance when adopting ConvE as the scoring function in three datasets. RGCN with DistMult performs similar to that with ConvE on FB15K-237. However, it dramatically outperforms RGCN with ConvE on WN18RRand NELL-995. This indicates that the choice of scoring functions has strong impact on the performance, and the impact is dataset-dependent. **(2)** Comparing CompGCN (or KBGAT) with RGCN on FB15K-237, even if the two methods adopt the same scoring function (either DistMult or ConvE), they still achieve quite different performance. On the WN18RRdataset, the observations are more involved. The two methods achieve similar performance when DistMult is adopted but behave quite differently with ConvE. Overall, these observations indicate that the scoring function is not the only factor impacting the model performance.

3.4.3 Loss Function Impact

In this subsection, we investigate the impact of the loss function on these three methods while fixing the scoring function and other experimental settings. As introduced in Section 3.3.2.3, in the original settings, CompGCN, RGCN and KBGAT adopt the BCE loss. The major difference in the loss function is that CompGCN and KBGAT utilize all negative samples while RGCN adopts a sampling strategy to randomly select 10 negative samples for training. For convenience, we use *w/o sampling* and *with sampling* to denote these two settings and investigate how these two settings affect the model performance.

3.4.3.1 Impact of negative sampling

To investigate the impact of negative sampling strategy, we also run CompGCN and KBGAT under the *with sampling* setting (i.e., using 10 negative samples as the original RGCN), and RGCN under the *w/o sampling* setting. The results are shown in Table 3.2, where we use "with" and "w/o" to denote these two settings. From Table 3.2, we observe that RGCN, CompGCN and KBGAT achieve stronger performance under the "*w/o sampling*" setting on three datasets. Specifically, the performance of CompGCN dramatically drops by 30.3% from 47.2 to 32.9 when adopting the sampling strategy, indicating that the sampling strategy significantly impacts model performance. Notably, only using 10 negative samples proves insufficient. Hence, we further investigate how the number of negative samples affects the model performance in the following subsection.

3.4.3.2 Impact of number of negative samples

In this subsection, we investigate how the number of negative samples affects performance under the "*with sampling*" setting for both methods. We run RGCN, CompGCN and KBGAT with varyinng numbers of negative samples. Following the settings of scoring functions as mentioned in Section 3.3.2.4., we adopt DistMult for RGCN and ConvE for CompGCN and KBGAT as scoring functions. Table 3.3 shows the results and #Neg is the number of negative samples. Note that in

Table 3.3, N denotes the number of entities in a KG, thus N differs across the datasets. In general, increasing the number of negative samples from 10 to a larger number is helpful for all methods. This partially explains why the original RGCN typically under-performs CompGCN and KBGAT. On the other hand, to achieve strong performance, it is not necessary to utilize all negative samples; for example, on `FB15K-237`, CompGCN achieves the best performance when the number of negative samples is 200; this is advantageous, as using all negative samples is more expensive. In short, carefully selecting the negative sampling rate for each model and dataset is important.

3.5 KGC without Message Passing

It is well known that MP is the key bottleneck for scaling MPNNs to large graphs [46, 147, 151]. Observation 1 suggests that the MP may be not helpful for KGC. Thus, in this section, we investigate if we can develop MLP-based methods (without MP) for KGC that can achieve comparable or even better performance than existing MPNN methods. Compared with the MPNN models, MLP-based methods enjoy the advantage of being more efficient during training and inference, as they do not involve expensive MP operations. We present the time complexity in Appendix B.8. The scoring and loss functions play an important role in MPNN-based methods. Likewise, we next study the impact of the scoring and loss functions on MLP-based methods.

Table 3.4 KGC results (%) of MLP-based methods with different combinations of scoring and loss functions. Both the scoring and loss functions impact the performance of MLP-based models.

		FB15K-237				WN18RR				NELL-995			
	#Neg	MRR	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10
DistMult	10	29.1±0.3	19.1±0.4	32.7±0.4	48.9±0.3	**44.0± 0.0**	39.5±0.5	**45.7± 0.3**	**51.9±0.5**	27.5±0.2	20.0±0.1	30.9±0.1	42.0±0.3
	50	31.3±0.2	21.2±0.3	35.4±0.3	51.1±0.1	42.5± 0.4	38.3±0.5	43.9±0.4	51.1±0.2	27.5± 0.4	19.4±0.6	31.7±0.3	42.7±0.1
	200	32.5±0.3	22.3±0.3	**37.1±0.3**	**52.0± 0.2**	41.5± 0.5	37.6±0.5	42.6± 0.6	49.6 ± 0.6	29.1±0.2	21.1±0.2	33.2±0.0	43.9 ±0.3
	0.5N	32.8±0.2	23.4±0.3	36.8± 0.1	50.7± 0.1	41.3±0.6	38.0±0.4	42.4±0.7	48.5±1.5	31.5± 0.2	23.9±0.2	35.5± 0.2	45.3±0.3
	N	32.7± 0.1	23.5 ±0.1	36.5±0.1	50.4± 0.1	41.4±0.2	38.4±0.2	42.2±0.2	47.7±0.2	31.0±0.1	23.4±0.3	34.8±0.3	45.1±0.4
	w/o	**33.4**±0.2	**24.5 ± 0.2**	36.6± 0.2	51.1± 0.2	43.3± 0.1	**39.9± 0.1**	44.6±0.2	50.7 ±0.9	**32.8±0.2**	**25.0±0.2**	**36.5 ± 0.3**	**47.7 ± 0.3**
ConvE	10	30.3 ±0.4	21.1 ±0.5	33.6 ±0.4	48.6 ±0.4	35.5 ±5.8	27.6 ± 8.4	40.5 ±3.7	49.3 ± 1.2	31.6 ±0.6	23.5 ±0.5	35.0 ± 0.7	47.2 ±0.6
	50	34.0± 0.3	24.5± 0.3	37.9± 0.2	52.6±0.2	42.1±0.1	36.3±0.2	45.0±0.1	52.4±0.1	37.0± 0.3	28.6± 0.4	41.4±0.3	53.1±0.1
	200	35.0±0.0	25.5±0.1	39.0± 0.1	53.4±0.1	44.2±0.3	39.9± 0.4	45.7± 0.3	52.6±0.1	38.9±0.3	30.8±0.4	43.3±0.4	**54.0±0.1**
	500	35.3±0.0	25.7±0.0	**39.2± 0.2**	53.6±0.2	44.5±0.3	40.6±0.4	45.7± 0.2	52.3±0.2	39.3±0.3	31.6±0.3	**43.5 ±0.4**	53.6±0.3
	0.5N	34.3±0.1	25.0± 0.2	38.1± 0.0	52.4±0.0	45.4±0.2	41.8±0.3	46.4±0.3	52.6±0.2	**40.0 ± 0.1**	**33.3± 0.2**	43.3±0.1	52.9±0.1
	N	34.0±0.1	24.8±0.2	37.7± 0.1	51.9±0.1	45.4±0.2	41.8±0.2	46.5±0.3	52.4 ± 0.1	39.7±0.2	33.0± 0.1	43.1±0.2	52.5± 0.1
	w/o	**35.5 ± 0.2**	**26.4±0.2**	38.9±0.2	**53.7±0.1**	**47.3± 0.1**	**43.7±0.2**	**48.8 ±0.1**	**54.4±0.1**	38.1 ±0.5	30.4±0.5	42.1±0.5	52.5±0.5

3.5.1 MLPs with various scoring and loss

We investigate the performance of MLP-based models with different combinations of scoring and loss functions. Specifically, we adopt DistMult and ConvE as scoring functions. For each

Table 3.5 KGC results (%) of the ensembled MLP-based methods, which outperform the MPNN-based models.

	FB15K-237				WN18RR				NELL-995			
	MRR	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10
CompGCN	35.5±0.1	26.4±0.1	39.0±0.2	53.6±0.3	47.2±0.2	43.7±0.3	48.5±0.3	54.0±0.0	38.1±0.4	30.4± 0.5	42.2±0.3	52.9± 0.1
RGCN	29.6± 0.3	19.1±0.5	34.0± 0.2	50.1±0.2	43.0±0.2	38.6±0.3	45.0±0.1	50.8±0.3	27.8±0.2	19.9±0.2	31.4±0.0	43.0±0.3
KBGAT	35.0±0.3	26.0±0.3	38.5± 0.3	53.1± 0.3	46.4±0.2	42.6±0.2	47.9±0.3	53.9±0.2	37.4±0.6	29.7±0.7	41.4± 0.8	52.0±0.4
MLP-best	35.5 ± 0.2	26.4±0.2	38.9±0.2	53.7±0.1	47.3± 0.1	43.7±0.2	48.8 ±0.1	54.4±0.1	40.0 ± 0.1	33.3± 0.2	43.3±0.1	52.9±0.1
MLP-ensemble	**36.9 ±0.2**	**27.5±0.2**	**40.8±0.2**	**55.4±0.1**	**47.7±0.3**	**43.9 ±0.4**	**48.9±0.1**	**55.4±0.1**	**41.7±0.2**	**34.7±0.2**	**45.1±0.0**	**55.2±0.1**

scoring function, we try both the *with sampling* and *w/o sampling* settings for the loss function. Furthermore, for the *with sampling* setting, we vary the number of negative samples. The results of MLP-based models with different combinations are shown in Table 3.4, which begets the following observations: **(1)** The results from Table 3.4 further confirm that the MP component is unnecessary for KGC. The MLP-based models can achieve comparable or even stronger performance than GNN models. **(2)** Similarly, the scoring and the loss functions play a crucial role in the KGC performance, though dataset-dependent. For example, it is not always necessary to adopt the *w/o* setting for strong performance: On the FB15K-237dataset, when adopting ConvE for scoring, the MLP-based model achieves comparable performance with 500 negative samples; on WN18RR, when adopting DistMult for scoring, the model achieves best performance with 10 negative samples; on NELL-995, when adopting ConvE for scoring, it achieves the best performance with $0.5N$ negative samples.

Given these observations, next we study a simple ensembling strategy to combine different MLP-based models, to see if we can obtain a strong and limited-complexity model which can perform well for various datasets, without MP. Note that ensembling MLPs necessitates training multiple MLPs, which introduces additional complexity. However, given the efficiency of MLP, the computational cost of ensembling is still acceptable.

3.5.2 Ensembling MLPs

According to Section 3.5.1, the performance of MLP-based methods is affected by the scoring function and the loss function, especially the negative sampling strategy. These models with various combinations of scoring function and loss functions can potentially capture important information from different perspectives. Therefore, an ensemble of these models could provide an opportunity to combine the information from various models to achieve better performance. Hence, we select some

MLP-based models that exhibit relatively good performance on the validation set and ensemble them for the final prediction. Next, we briefly describe the ensemble process. These selected models are individually trained, and then assembled together for the inference process. Specifically, during the inference stage, to calculate the final score for a specific triplet (h, r, t), we utilize each selected model to predict a score for this triplet individually and then add these scores to obtain the final score for this triplet. The final scores are then utilized for prediction. In this work, our focus is to show the potential of ensembling instead of designing the best ensembling strategies; hence, we opt for simplicity, though more sophisticated strategies could be adopted. We leave this to future work.

We put the details of the MLP-based models we utilized for constructing the ensemble model for these three datasets in Appendix B.9. The results of these ensemble methods are shown in Table 3.5, where we use MLP-ensemble to generally denote the ensemble model. Note that MLP-best in the table denotes the best performance from individual MLP-based methods from Table 3.4. From the table, we can clearly observe that MLP-best can achieve comparable or even slightly better performance than MPNN-based methods. Furthermore, the MLP-ensemble can obtain better performance than both the best individual MLP-based methods and the MPNN-based models, especially on `FB15K-237`and `NELL-995`. These observations further support that the MP component is not necessary. They also indicate that these scoring and loss functions are potentially complementary to each other, and as a result, even the simple ensemble method can produce better performance.

3.6 Discussion

Key Findings: **(1)** The MP component in MPNN-based methods does not significantly contribute to KGC performance, and MLP-based methods without MP can achieve comparable performance; **(2)** Scoring and the loss function design (i.e. negative sampling choices) play a much more crucial role for both MPNN-based and MLP-based methods; **(3)** The impact of these is significantly dataset-dependent; and **(4)** Scoring and the loss function choices are complementary, and simple strategies to combine them in MLP-based methods can produce better KGC performance.

Practical Implications: **(1)** MLP-based models do not involve the complex MP process and thus they are more efficient than the MPNN-based models [147]. Hence, such models are more scalable and can be applied to large-scale KGC applications for practical impact; **(2)** The simplicity and scalability of MLP-based models make ensembling easy, achievable and effective (Section 3.5.2); and **(3)** The adoption of MLP-based models enables us to more conveniently apply existing techniques to advance KGC. For instance, Neural Architecture Search (NAS) algorithms [158] can be adopted to automatically search better model architectures, since NAS research for MLPs is much more extensive than for MPNNs.

Implications for Future Research: (1) Investigating better designs of scoring and loss functions are (currently) stronger levers to improve KGC. Further dedicated efforts are required for developing suitable MP operations in MPNN-based models for this task; **(2)** MLP-based models should be adopted as default baselines for future KGC studies. This aligns with several others which suggest the underratedness of MLPs for vision-based problems [68, 107]; **(3)** Scoring and loss function choices have complementary impact, and designing better strategies to combine them is promising; and **(4)** Since KGC is a type of link prediction, and many works adopt MPNN designs in important settings like ranking and recommendations [139, 28, 116], our work motivates a pressing need to understand the role of MP components in these applications.

3.7 Conclusion

In this paper, we surprisingly find that the MLP-based models are able to achieve competitive performance compared with three MPNN-based models (i.e., CompGCN, RGCN and KBGAT) across a variety of datasets. It suggests that the message passing operation in these models is not the key component to achieve strong performance. To explore which components potentially contribute to the model performance, we conduct extensive experiments on other key components such as scoring function and loss function. We found both of them play crucial roles, and their impact varies significantly across datasets. Based on these findings, we further propose ensemble methods built upon MLP-based models, which are able to achieve even better performance than MPNN-based models.

CHAPTER 4

LANGUAGE MODEL-ENHANCED LINK PREDICTION.

Session-based recommendation systems have recently attracted considerable interest, aiming to provide personalized recommendations by analyzing users' historical activities within sessions. While ID-based methods have shown considerable promise, they often struggle with long-tail items and fail to consider other forms of information, such as valuable textual information. To incorporate text information, various approaches have been proposed, generally employing a naive fusion framework. Interestingly, it appears that combining ID and text using the naive fusion does not consistently surpass the performance of the best single-modality approach. Further exploration indicates a potential imbalance issue in the naive fusion method, where the ID tends to overshadow the text and the text is undertrained. This issue indicates that the naive fusion method might not be as effective in combining ID and text as once believed. To address these challenges, we introduce an alternative training approach, AlterRec, which separates the training of ID and text to prevent the previously observed imbalance issue. AlterRec also designs an innovative strategy to enhance the interaction between the two modalities, facilitating mutual interaction and more effective text integration. Extensive experiments demonstrate the effectiveness of AlterRec in session-based recommendation.

4.1 Introduction

In recent years, predicting the next item in user-item interaction sequences, such as clicks or purchases, has gained increasing attention [126, 58, 81, 37]. This practice is prevalent across various online platforms, including e-commerce, search engines, and music/video streaming sites. These sequences are created during user-item interactions in sessions. They encode user preferences which are dynamic and evolve over time [106]. Moreover, in many systems, only the user's behavior history during an ongoing session is accessible. Therefore, analyzing interactions in active sessions becomes essential for real-time recommendations. This need has spurred the development of session-based recommendations [126, 37], which utilizes the sequential patterns in a session to understand and predict the latest user preferences.

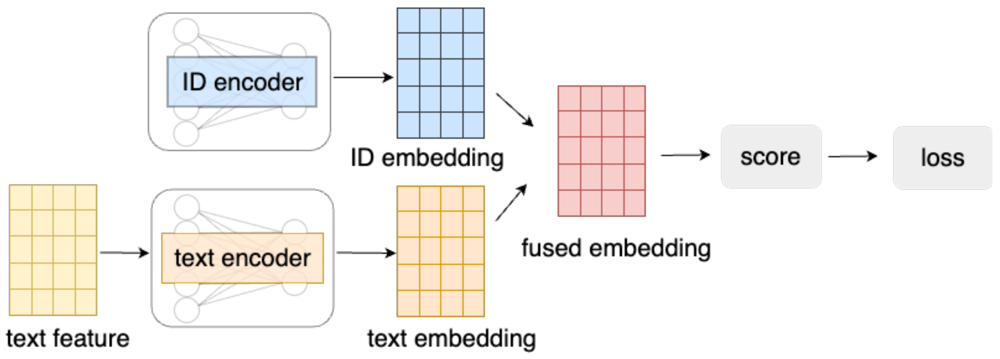


Figure 4.1 An illustration of a naive fusion framework. It has two encoders to extract the ID and text information.

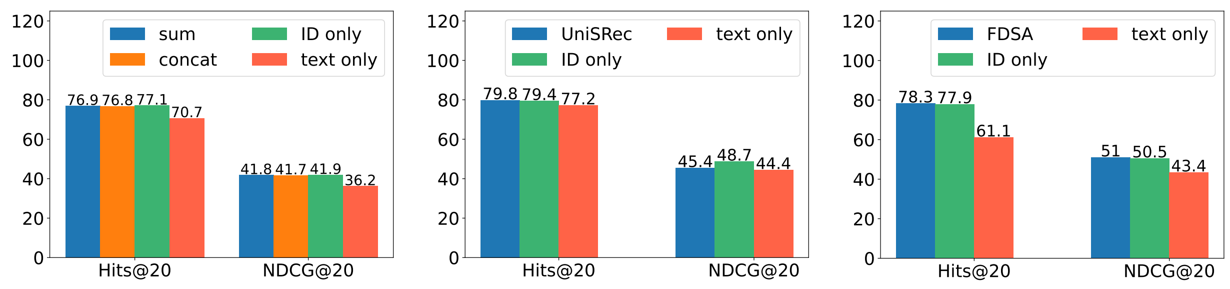
In this domain, ID-based methods [48, 103, 126] have become the predominant approach, significantly influencing the recommendation paradigm [142, 64]. These methods involve assigning unique ID indexes to users and items, transforming them into vector representations. Their popularity stems from their simplicity and effectiveness across various applications [64]. Despite their proven effectiveness, these methods still have limitations. One drawback is their heavy reliance on the ID-based information. They tend to overlook other forms of valuable data, notably rich text information. This exclusion of textual data can result in less informative representations. It can be problematic in scenarios with limited interactions between users and items. However, most items experience sparse interactions, known as long-tail items [82], which presents a challenge for these methods.

Recognizing these limitations, there has been a shift towards integrating text data for recommendations. The surging volume of text data emphasizes the crucial role of text in various domains, such as news recommendation [57, 123] and e-commerce [45]. These systems aim to match user preferences by processing and encoding textual content like user reviews, product descriptions, and news articles. Recent trends indicate an increasing reliance on language models [50, 11, 119, 35] for extracting semantic information due to their exceptional ability to encode text effectively. This progress has sparked considerable interest in enhancing recommendation beyond traditional user-item interaction data.

The prevailing approach in current literature for combining ID and text typically employs a **naive**

fusion framework [38, 148, 119], as shown in Figure 4.1. It involves generating embeddings from ID and text encoders, merging them, and using this for loss computation. However, our preliminary study in section 4.3.2 reveals that the naive fusion may not be as effective as previously believed in combining ID and text information. 1) Notably, it shows that independent training on just the ID information can yield performance comparable to, or even better than, the naive fusion model. This implies that the naive fusion model may not necessarily enhance, and could potentially reduce the overall performance. This finding aligns with the studies in multi-modal learning [40, 115, 23], which indicates that the fusion of multiple modalities doesn't always outperform the best single modality. 2) We further explore one naive fusion implementation as an example to have a deeper understanding of this finding. The exploration suggests a potential imbalance issue: the model heavily relies on the ID component, while the text component appeared undertrained. This imbalance implies that the unexpected finding might be a result of the naive fusion framework's inability to balance the contributions of the two types of information effectively, thereby hindering optimal overall performance.

The imbalance issue identified in the naive fusion models significantly hinders the accurate integration of textual data. Despite increased efforts to integrate textual content, these methods often fail to effectively capture essential semantic information. It results in a considerable loss of valuable information. This realization shifts our focus towards independent training, which does not exhibit this issue. However, independent training overlooks the potential for ID and text to provide complementary information that could be mutually learning. To address these challenges, we propose a novel **Alter**native training strategy to combine the ID and text components for session-based **Rec**ommendation (**AlterRec**). This approach separates the training of ID and text and thereby avoiding the imbalance issue. Additionally, it goes beyond simple independent training by enabling implicit interactions between these two modalities, thereby allowing them to inform and learn from each other. We conduct comprehensive experiments to validate the superior effectiveness of AlterRec over a variety of baselines in real-world datasets.



(a) NFRec (b) UnisRec (c) FDSA

Figure 4.2 Session-based recommendation results (%) on the Amazon-French dataset. We compare the models combing ID and text against models trained independently on either ID or text information alone.

4.2 Related Work

ID-based Methods. These methods [36] convert each user or item into a vector representation using unique ID indices. More recent advancements have seen the adoption of sophisticated architectures as encoders. For instance, SASRec [48] and BERT4Rec [103] employ the Transformer architecture to delineate user preferences within sequences. SR-GNN [126] and HG-GNN [81] construct graphs from the user-item interaction data to capture complex patterns across multiple sessions. However, these methods overlook additional valuable text information, potentially leading to less informative representations.

Text-Integrated Methods. These methods combine the text information to perform recommendations. For instance, FDSA [148] leverages concatenation and S^3-Rec [153] uses self-supervised tasks to combining textual information. UniSRec [38] employs the BERT model and LLM2BERT4Rec [35] uses the text feature from by the large language model (LLMs) as initialization. RLMRec [90] and LLMRec [119] both use LLMs for generating user/item profiles. Among the methods discussed, the majority follows the naive fusion framework [148, 38, 119], which may not effectively incorporate text as identified in the section 4.3.2.

4.3 Preliminaries

4.3.1 Session-based Recommendation

Consider a set of users and items, denoted as $\mathcal{U}$ and $\mathcal{V}$, respectively. Let $\mathcal{S}$ denote the set of user-item interaction sequences (or sessions). We use $\mathbf{s} = \{s_1, s_2, , ..., s_n\} \in \mathcal{S}$ to represent one of the sequences, where $s_i \in \mathcal{V}$ is the i-th item interacted with the same user in session $\mathbf{s}$, and n is the total number of interactions. The number of users and items are represented by $|\mathcal{U}|$ and $|\mathcal{V}|$, respectively. Each item i in $\mathcal{V}$ is associated with text information, such as product descriptions, titles, or taxonomies, denoted by $t_i = \{w_1, w_2, ..., w_c\}$. Here, each word w_j belongs to a shared vocabulary, and c represents the truncated length of the text. Given an item sequence, the objective of session-based recommendation is to predict the next item in the current sequence. Formally, this involves generating a ranking list $\mathbf{y}_s$ for all candidate items, where $\mathbf{y_s} = [y_{\mathbf{s},1}, ..., y_{\mathbf{s},|\mathcal{V}|}]$ and each $y_{\mathbf{s},i}$ is a score indicating the likelihood of item i being the next interacted item given a session $\mathbf{s}$.

The Naive Fusion Framework. In session-based recommendation tasks, ID-based and text-based information can be combined to potentially improve the overall performance. The majority of existing methods employ a naive fusion approach combined with a joint training strategy [38, 148, 119], as illustrated in Figure 4.1. Specifically, this approach involves generating two types of embeddings $\mathbf{X}^{ID}$ and $\mathbf{X}^{text}$ using ID and text encoders, respectively. These two embeddings which can be item-level or session-level, are then merged into a final embedding $\mathbf{Z}$, through methods such as summation or concatenation. This final embedding is used to calculate a relevance score between a given session and the candidate item, estimating the likelihood of the item being the next choice. Throughout this paper, the term **naive fusion** is used to refer to this framework. Notably, existing methods such as UniSRec [38], FDSA [148], and LLMRec [119] follow this approach. We implemented a naive fusion method which is named NFRec and more details are in section C.1 in the Appendix.

4.3.2 Preliminary Study

In this subsection, motivated by multi-modal learning [115, 40, 83], we conduct a preliminary study to investigate potential challenges in combining ID and text information for session-based recommendation, aiming to inspire more effective integration strategies.

In the naive fusion framework, the ID and text can be treated as two different types of modality that work together to improve the overall performance. However, studies in the multi-modal learning [115, 23, 40, 83] reveals **a phenomenon: fusing two modalities does not usually outperform the best single modality trained independently.** In other words, combining modalities may not enhance, and could potentially reduce, overall performance. Various studies have focused on this phenomenon, offering analysis from different perspectives, such as greedy learning [124], modality competition [40] and modality laziness [24]. To effectively merge ID and text information for session-based recommendations, we conduct an investigation to first verify the presence of this phenomenon and then explore its underlying causes.

4.3.2.1 Naive Fusion vs. Independent Training

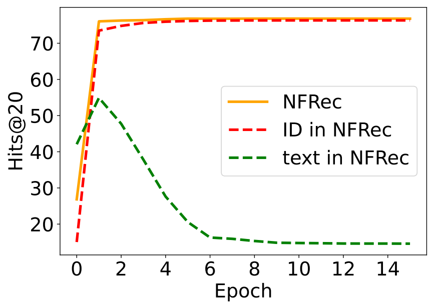
To examine if the phenomenon mentioned above exists in session-based recommendation, this investigation aims to compare the performance of naive fusion models including our implementation NFRec, UniSRec [38] and FDSA [148] against their corresponding two single modality models (ID and text) that are trained independently. To ensure a fair comparison, we employ the same ID/text encoder, scoring function, and loss function across both naive fusions and independent training frameworks. More details are given in section C.1 in the Appendix.

The results on Amazon-French (detailed in the section 4.5.1.1) are presented in Figure 4.2 , where “ID only” and “text only” denote the respective ID and text only models that are trained independently, “sum” and “concat” represent our naive fusion implementations using summation and concatenation respectively to combine ID and text. We employ two widely used metrics Hits@20 and NDCG@20, where higher scores indicate better performance. We also investigated on the HD dataset in Figure C.1 in the Appendix which shows similar phenomenon. We have the following observations:

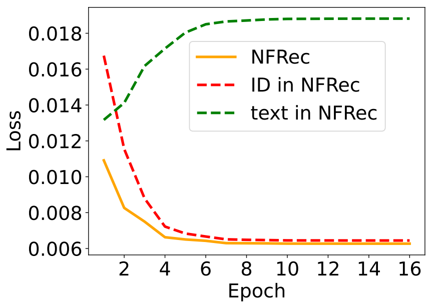
Observation 2 *Training solely with ID information independently can often achieve performance comparable to, or even better than, naively fusing both ID and text. It indicates the ineffectiveness of naive fusion for combining ID and text.*

Observation 3 *The text only model generally results in the worst performance, often exhibiting a substantial performance gap compared to the ID only approach.*

The first observation aligns with findings in multi-modal learning studies [83, 40], indicating a similar phenomenon. This suggests that the integration of ID and text is not as effective as expected



(a) Amazon-French



(b) Amazon-French

Figure 4.3 Test performance in terms of Hits@20 (%) and training loss comparison on the Amazon-French dataset.

in the session-based recommendation. To gain a more comprehensive understanding of this issue, we will delve into NFRec, which are elaborated in the following subsection.

4.3.2.2 Exploration of NFRec

In our exploration, we aim to understand how ID and text components perform under NFRec, shedding light on why their combination in a naive fusion framework may not yield the expected improvement. To this end, we take NFRec applying concatenation for fusing ID and text information as one example. NFRec can be conceptually divided into two segments: the ID component and the text component (further details are in section C.2 in the Appendix). Test performance and training loss on the Amazon-French dataset are shown in Figure 4.3, with the components labeled as “ID in NFRec" and “text in NFRec." We observe a similar phenomenon on the HD dataset, as shown in Figure C.2 in the Appendix.

Figure 4.3 reveals a significant imbalance issue in NFRec: the performance and loss of the ID component are almost overlapping with those of NFRec, which indicates a heavy reliance on the ID component. The ID dominates the overall performance and loss, and the text component has limited contributions. This suggests that the Observation 1 may stem from the nature of naive fusion. Specifically, it appears incapable of balancing the modalities to achieve optimal overall performance and tends to overly depend on the stronger ID modality (as noted in the Observation 2). Supporting this hypothesis is from various studies [83, 40, 115, 124] in multi-modal learning which offer empirical and theoretical insights. Further investigation to identify more concrete causes of

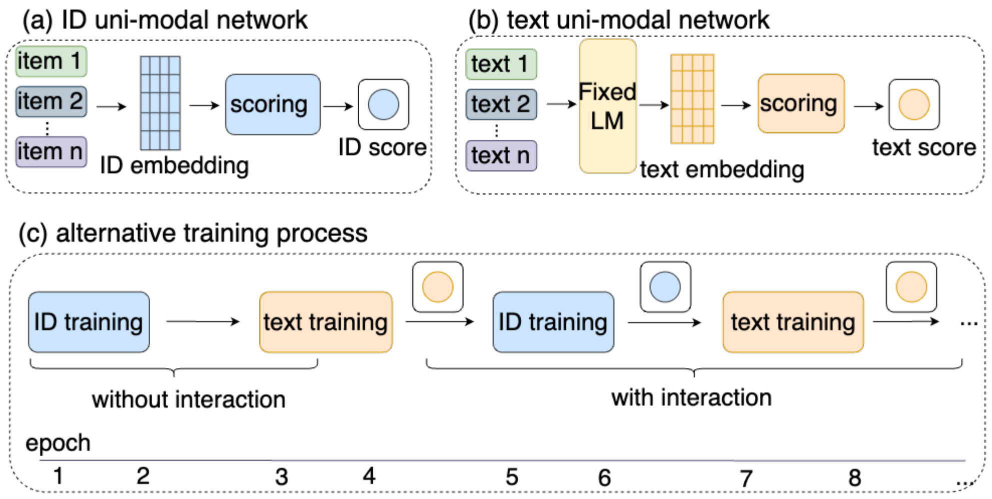


Figure 4.4 An Overview of AlterRec. (a), (b): two key components – the ID and text uni-modal networks. (c): These networks are trained alternately, learning from each other through predictions generated by the other network.

this phenomenon is designated as one future work.

4.4 Framework

Having identified the potential imbalance issue with the naive fusion framework in the previous section, we explore to combine ID and text information by training them separately. However, simply training ID and text independently may not fully exploit their potential to provide complementary information. To address these challenges, we introduce AlterRec, a novel alternative training method, as illustrated in Figure 4.4. The model comprises two key components: the ID and text uni-modal networks, designed to capture ID and semantic information respectively. We employ the predictions from one network as training signals for the other, facilitating interaction and mutual learning through these predictions. AlterRec separates the training of ID and text, effectively avoiding the imbalance issue. Moreover, it goes beyond independent training by facilitating interaction between ID and text, enabling them to learn mutually beneficial information and incorporate the text more effectively.

4.4.1 ID and Text Uni-modal Networks

The ID and text unimodal networks share similar architectures. Each has respective ID/text encoders to generate ID and text embeddings. Based on these embeddings, a scoring function is adopted to calculate the relevance between given session and candidate items. We first introduce two encoders, and then show how to define the scoring function.

4.4.1.1 ID Encoder

The ID encoder is designed to create a unique embedding for each item based on its ID index. This is achieved using an ID embedding matrix $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$, where d is the size of the embedding. Each row, $\mathbf{X}_i$, corresponds to the ID embedding of item i. Notably, this matrix is a learnable parameter and updated during the optimization.

4.4.1.2 Text Encoder

The text encoder is designed to extract textual information from items. Leveraging the advanced language modeling capabilities, we utilize the Sentence-BERT [89] in this work. For item i, represented by a text sequence $t_i = \{w_1, w_2, ..., w_c\}$, the Sentence-BERT processes the input sentence to generate token embeddings and output a comprehensive embedding for the entire sentence. We further use a MLP [20] to transform the embeddings from the Sentence-BERT into a $d-$dimensional matrix. Formally:

$$\mathbf{H}_i = \text{MLP}(\text{SBERT}(w_1, w_2, ..., w_c)), \tag{4.1}$$

where $\mathbf{H} \in \mathbb{R}^{|\mathcal{V}| \times d}$ and each row $\mathbf{H}_i$ corresponds to the text embedding of item i. Considering practical constraints, we fix the language model which isn't updated during the optimization process due to the high training cost.

4.4.1.3 Scoring Function

In the context of session-based recommendation, our objective is to predict the next item in a sequence of items interacted with the same user, denoted as $\mathbf{s} = \{s_1, s_2, ..., s_n\}$. To accomplish this, we generate a prediction score, $y_{\mathbf{s},j}$, for each candidate item j. These scores are then used to rank all candidate items, with the top-ranked item predicted as the next item. The process begins with obtaining the session embedding $\mathbf{q_s} \in \mathbb{R}^d$ for session $\mathbf{s}$, which encodes the user's interaction behavior. The relevance between the session embedding and each candidate item's embedding is calculated and used as the score for that item.

Session&Item Relevance. The session embedding is derived from the embeddings of the items within the session, either ID or text embeddings. We use the ID embedding as an example since it's similar for the text embedding. Formally, we define a function g to generate the session embedding: $\mathbf{q_s} = g(\mathbf{X}_{s_1}, \mathbf{X}_{s_2}, \mathbf{X}_{s_n})$. A simple yet effective approach is the **mean function** which calculates the mean of the item embeddings. Alternatively, the **Transformer** architecture [111] can be employed to capture item-item transition patterns. When using transformer, we use the embedding of the last item in the sequence as the session embedding, as it encapsulates information from all items. The session embedding can be derived using either of these functions based on their empirical performance. We determine the relevance between the session and each candidate item using vector multiplication: $y^{ID}_{\mathbf{s},i} = \mathbf{X}_i^T \mathbf{q_s}$. Similarly, we can get the score $y^{text}_{\mathbf{s},i} = \mathbf{H}_i^T \hat{\mathbf{q}}_\mathbf{s}$ based on the text embedding, where $\hat{\mathbf{q}}_\mathbf{s} = g(\mathbf{H}_{s_1}, \mathbf{H}_{s_2}, \mathbf{H}_{s_n})$.

4.4.2 Alternative Training

The ID and text data offer different information. Our goal is to facilitate their interaction, enabling mutual learning and thereby enhancing overall performance. To this end, we propose an alternative training strategy to use predictions from one uni-modal network to train the other network. These predictions encode information of one modality, allowing one network to learn information from the other. We leverage the predictions from one modality to the other in two aspects. 1) First, we select top-ranked items as augmented positive training samples. These items with top scores are likely very relevant to the current session from the perspective of one modality that could provide more training signals for the other modality especially for items with fewer interactions. 2) Second, we choose other high-scored items as negative samples. These items are ranked higher but not the most relevant ones for one modality and we aim to force the other modality to distinguish them from positive samples. Such negative samples are much harder to be distinguished compared to those from traditional random sampling [92]. Thus, we refer to them as hard negative samples in this work.

Hard Negative Samples. To illustrate, we use the predictions from the ID uni-modal network to train the text uni-modal network as an example. We first generate predictions from the ID uni-modal network for a given session $\mathbf{s}$. Then we rank the scores of all candidate items in descending order $r_{\mathbf{s}}^{ID} = \text{argsort}(y_{\mathbf{s},1}^{ID}, y_{\mathbf{s},2}^{ID}, ..., y_{\mathbf{s},|\mathcal{V}|}^{ID})$ where $r_{\mathbf{s}}^{ID}$ denotes the sequence of ID indices corresponding to the sorted scores. We select items ranked from k_1 to k_2, represented as $r_{\mathbf{s}}^{ID}[k_1 : k_2]$, as the hard negative samples for training the text uni-modal network. It enables the text uni-modal network to learn from the patterns identified by the ID uni-modal network. These hard negatives play a crucial role in defining the loss function. For a given session $\mathbf{s}$ with s_t as the target item, we use the cross entropy as the loss function by following most of the related works [38, 126, 81]:

$$L^{text} = -\sum_{\mathbf{s}\in\mathcal{S}} \log(f(y_{\mathbf{s},s_t}^{text})) \tag{4.2}$$

where f is the Softmax function applied over the target item s_t and the negative samples in $r_{\mathbf{s}}^{ID}[k_1 : k_2]$.

Similarly, we can use the hard negative sample $r_{\mathbf{s}}^{text}[k_1 : k_2]$ derived from the text uni-modal network by sorting the scores and identifying the ranking ID index $r_{\mathbf{s}}^{text} = \text{argsort}(y_{\mathbf{s},1}^{text}, y_{\mathbf{s},2}^{text}, ..., y_{\mathbf{s},|\mathcal{V}|}^{text})$. We define the loss function to train the ID uni-modal network as follows:

$$L^{ID} = -\sum_{\mathbf{s}\in\mathcal{S}} \log(f(y_{\mathbf{s},s_t}^{ID})) \tag{4.3}$$

where the Softmax function is applied over the target item s_t and the negative samples in $r_{\mathbf{s}}^{text}[k_1 : k_2]$.

Positive Sample Augmentation. To train the text uni-modal network, we utilize $r_{\mathbf{s}}^{ID}[1 : p]$ as additional positive samples which serves as ground-truth target items. Similarly, $r_{\mathbf{s}}^{text}[1 : p]$ is used as supplementary positive samples for training the ID uni-modal network. Typically, we set $p < k_1$. Accordingly, the losses in Eq. (4.2) and (4.3) are modified as follows:

$$L_a^{text} = -\sum_{\mathbf{s}\in\mathcal{S}} \left(\log(f(y_{\mathbf{s},s_t}^{text})) + \beta * \sum_{s_k \in r_{\mathbf{s}}^{ID}[1:p]} \log(f(y_{\mathbf{s},s_k}^{text})) \right) \tag{4.4}$$

$$L_a^{ID} = -\sum_{\mathbf{s}\in\mathcal{S}} \left(\log(f(y_{\mathbf{s},s_t}^{ID})) + \beta * \sum_{s_k \in r_{\mathbf{s}}^{text}[1:p]} \log(f(y_{\mathbf{s},s_k}^{ID})) \right) \tag{4.5}$$

Here, β is a parameter to adjust the importance of the augmented samples. Note that within each network, these augmented samples are paired with the same corresponding hard negative samples as the target item s_t.

Algorithm 4.1 Alternative Training

```
Require: User-item interaction set $\mathcal{S}$, epoch number using random negatives $m_{random}$, maximum
    epoch number $m_{max}$, gap epoch number $m_{gap}$
Ensure: Converged models $\theta^{ID}$, $\theta^{text}$
 1: Random initialize two uni-modal networks $\theta^{ID}$, $\theta^{text}$
 2: for i = 1, 2, . . . , $m_{random}$ do
 3:     Train $\theta^{ID}$ using random negatives
 4: end for
 5: for i = 1, 2, . . . , $m_{random}$ do
 6:     Train $\theta^{text}$ using random negatives
 7: end for
 8: for i = 0, 1, . . . , $m_{max} - 2 * m_{random}$ do
 9:     if $i \mod (2 * m_{gap}) < m_{gap}$ then
10:         Compute loss in Eq. (4.3)
11:         Update $\theta^{ID} : \theta^{ID} \leftarrow \theta^{ID} - \alpha\nabla L^{ID}$
12:     else
13:         Compute loss in Eq. (4.2)
14:         Update $\theta^{text} : \theta^{text} \leftarrow \theta^{text} - \alpha\nabla L^{text}$
15:     end if
16: end for
```

Training Algorithm. This algorithm focuses on facilitating the interaction between two networks, and we use the Figure 4.4(c) as a more straightforward illustration. We present the pseudo code in Algorithm 4.1. The parameters within the ID and text uni-modal networks are denoted as θ^{ID} and θ^{text}, respectively. As indicated in line 1, both networks are randomly initialized. The training process consists of two stages. **1) Initially**, due to the lower quality of the learned embeddings, we don't employ interaction between two networks. Thus, we apply random negative samples during the first m_{random} epochs, as indicated in line 2-7. This involves replacing the hard negatives in Eq. (4.2) and Eq. (4.3) with randomly selected negatives with equal number. **2) Subsequently**, we shift to training with hard negatives. We start by training the ID uni-modal network using hard negatives derived from the text uni-modal network, as described in lines 9 to 11. After m_{gap} epochs, the training focus shifts to the text uni-modal network, which is trained using hard negatives from the ID uni-modal network, as shown in lines 12-15. Following another m_{gap} epochs, we resume training the ID uni-modal network and repeat this alternating process. Notably, for AlterRec_aug, we replace the loss function in line 10 and 13 as Eq. (4.5) and Eq. (4.4) respectively. This approach ensures that each network continually learns from the other, thereby potentially improving overall performance.

Upon convergence of both networks, we generate a final relevance score by combining the relevance scores from each network and weighting their contributions. This score is used during the **inference stage** and is defined as:

$$y_{\mathbf{s},i} = \alpha * y_{\mathbf{s},i}^{ID} + (1 - \alpha) * y_{\mathbf{s},i}^{text} \tag{4.6}$$

Here, $y_{\mathbf{s},i}$ is the final score for the candidate item i given session $\mathbf{s}$, and α is a pre-defined parameter.

Table 4.1 Data statistic of the session datasets. The Amazon-M2 datasets don't involve users. #Train, #Val, and #Test denote the number of sessions in the train, validation, and test.

Dataset	#User	#Item	#Train	#Val	#Test
HD	145,750	39,114	182,575	2,947	5,989
Amazon-Spanish	-	38,888	75,098	7,900	6,237
Amazon-French	-	40,258	96,245	10,507	8,981
Amazon-Italian	-	45,559	102,923	11,102	10,158

4.5 Experiment

In this section, we conduct comprehensive experiments to validate the effectiveness of AlterRec. In the following, we will introduce the experimental settings, followed by the results and their analysis.

4.5.1 Experimental Settings

4.5.1.1 Datasets

We adopt two real-world session recommendation datasets including textual data and the data statistic is presented in Table 4.1. **HD**: It is from an e-commerce company that is derived from user purchase logs on its website. **Amazon-M2** [45]: It's a multilingual dataset. For the purpose of this study, which does not focus on multilingual data, we extracted unilingual sessions to create individual datasets for three languages: Spanish, French, and Italian. They are denoted as **Amazon-Spanish**, **Amazon-French**, and **Amazon-Italian**, respectively.

The **HD** dataset is a sampled dataset of purchase logs from the Home Depot's website. We include sessions where all items have textual data, i.e., titles, descriptions, and taxonomy. Items in each session is interacted by the same user, who may have engaged in several sessions at distinct timestamps. For the purposes of validation and testing, we select the most recent sessions from different users. Specifically, 10% of these sessions are designated for validation and 20% for testing, with the remainder allocated to training sessions. Typically, the sessions in validation appear after those in the training set, and the sessions for testing appear after those in the validation set. For the three **Amazon-M2** datasets, since there is no original validation set, we use about 10% of the training set to create a validation set.

4.5.1.2 Baselines

In our study, we refer to the model without augmentation as **AlterRec** and the augmented version as **AlterRec_aug**. We include several baseline methods: **CORE** [37], **SASRec** [48], **BERT4Rec** [103], **SR-GNN** [126], and **HG-GNN** [81] as ID-based methods. Text-integrated methods include **LLM2BERT4Rec** [35], **UniSRec** [38], **FDSA** [148], and **S^3-Rec** [153]. Notably, **UniSRec (FHCKM)** refers to the model pretrained on the FHCKM dataset [38], while **UniSRec** in this work denotes the model pretrained on our datasets (HD and Amazon-M2). LLM2BERT4Rec uses BERT4Rec as a backbone model, and we also test **LLM2SASRec**, which uses SASRec. To ensure fairness, each baseline method uses the same input features as AlterRec, except for UniSRec (FHCKM), which is pretrained with fixed dimension sizes.

4.5.1.3 Settings

Empirically, for the HD dataset, we use the mean function for ID session embedding and Transformer for text session embedding. For Amazon-M2, Transformer is used for both ID and text session embeddings. Model performance is evaluated using Hits@K and NDCG@K, with K set to 10 and 20. Higher scores indicate better performance. We set the parameters as follows: $m_{random} = 2, m_{gap} = 2, m_{max} = 30$, $\alpha = 0.5$, $\beta = 0.5$, $p = 5$. Additionally, for HD, we set $k_1 = 6$, $k_2 = 2000$, and for the Amazon-M2 datasets, $k_1 = 20$, $k_2 = 20000$ are used.

We search the learning rate in $\{0.01, 0.001\}$ and dropout in $\{0.1, 0.3, 0.5\}$, and we set hidden dimension as 300, and number of Transformer layer to be 2, for all models. The test results we report are based on the model that achieves the best performance during the validation phase. For text feature extraction in the HD dataset, we utilize Sentence-BERT with the all-MiniLM-L6-v2 model[1]. In contrast, for the three Amazon-M2 datasets, we employ Sentence-BERT with the distiluse-base-multilingual-cased-v1 model[2], due to its proficiency in handling multiple languages including Spanish, French, and Italian. For each item in the HD dataset, we use title, description, and taxonomy as the textual data. For the Amazon-M2 datasets, we use the title and description as textual data. All baseline methods employ the cross entropy as loss function and are implemented based on the RecBole [3]. All the experiments except for the industrial setting are conducted on the NVIDIA L4 GPUs with 24Gb.

Table 4.2 Performance Comparison (%) on the HD and three Amazon-M2 datasets. All reported results are mean and standard deviation over three seeds. The best results are highlighted in bold, and the second-best results are underlined.

	HD				Amazon-Spanish			
	Hits@10	Hits@20	NDCG@10	NDCG@20	Hits@10	Hits@20	NDCG@10	NDCG@20
SASRec	33.58 ± 0.27	40.93 ± 0.14	18.23 ± 0.06	20.09 ± 0.06	70.95 ± 0.32	80.46 ± 0.32	44.88 ± 0.33	47.29 ± 0.34
BERT4Rec	26.06 ± 0.26	31.85 ± 0.45	15.61 ± 0.3	17.08 ± 0.35	64.6 ± 0.13	74.0 ± 0.33	44.6 ± 0.16	46.98 ± 0.18
SRGNN	30.09 ± 0.07	36.0 ± 0.19	15.31 ± 0.13	15.73 ± 0.13	67.02 ± 0.29	76.37 ± 0.12	46.75 ± 0.33	49.12 ± 0.26
HG-GNN	33.17 ± 0.13	40.72 ± 0.20	18.27 ± 0.49	20.19 ± 0.51	N/A	N/A	N/A	N/A
CORE	37.04 ± 0.11	44.73 ± 0.06	19.86 ± 0.14	21.81 ± 0.14	71.83 ± 0.15	81.14 ± 0.17	41.05 ± 0.06	43.41 ± 0.08
UnisRec (FHCM)	36.03 ± 0.12	43.67 ± 0.06	20.14 ± 0.79	22.08 ± 0.77	72.15 ± 0.01	81.3 ± 0.02	44.87 ± 0.1	47.2 ± 0.1
UnisRec	34.56 ± 0.23	42.19 ± 0.16	19.01 ± 0.08	20.92 ± 0.08	72.33 ± 0.06	81.42 ± 0.16	45.51 ± 0.05	47.82 ± 0.06
FDSA	32.1 ± 0.34	39.11 ± 0.2	20.44 ± 0.1	22.21 ± 0.06	70.55 ± 0.24	79.84 ± 0.08	49.83 ± 0.15	52.18 ± 0.13
S^3-Rec	26.69 ± 0.1	33.04 ± 0.34	16.01 ± 0.14	17.62 ± 0.11	69.61 ± 0.4	78.85 ± 0.62	47.25 ± 0.45	49.6 ± 0.4
LLM2SASRec	34.12 ± 0.29	42.13 ± 0.18	18.69 ± 0.26	20.72 ± 0.22	71.55 ± 0.06	80.68 ± 0.12	48.45 ± 0.15	50.77 ± 0.17
LLM2BERT4Rec	29.51 ± 0.35	37.3 ± 0.33	16.25 ± 0.3	18.22 ± 0.3	66.47 ± 0.2	76.95 ± 0.27	40.29 ± 0.32	42.95 ± 0.35
AlterRec	38.25 ± 0.14	46.31 ± 0.11	20.72 ± 0.06	**22.76 ± 0.06**	72.41 ± 0.17	**81.49 ± 0.09**	**50.59 ± 0.14**	**52.9 ± 0.12**
AlterRec_aug	**38.46 ± 0.1**	**46.37 ± 0.08**	**20.74 ± 0.05**	22.75 ± 0.03	**72.47 ± 0.19**	81.45 ± 0.04	50.58 ± 0.02	52.86 ± 0.05

	Amazon-French				Amazon-Italian			
	Hits@10	Hits@20	NDCG@10	NDCG@20	Hits@10	Hits@20	NDCG@10	NDCG@20
SASRec	69.2 ± 0.15	78.4 ± 0.1	44.89 ± 0.43	47.23 ± 0.44	68.25 ± 0.08	78.37 ± 0.06	43.24 ± 0.18	45.81 ± 0.18
BERT4Rec	63.01 ± 0.11	72.47 ± 0.15	43.84 ± 0.04	46.24 ± 0.07	62.24 ± 0.26	72.38 ± 0.13	42.42 ± 0.15	44.99 ± 0.11
SRGNN	65.61 ± 0.09	74.93 ± 0.09	46.27 ± 0.1	48.64 ± 0.08	65.62 ± 0.26	75.2 ± 0.15	44.85 ± 0.17	47.28 ± 0.15
CORE	69.93 ± 0.02	79.32 ± 0.1	39.4 ± 0.05	41.79 ± 0.07	69.42 ± 0.12	79.4 ± 0.1	39.27 ± 0.05	41.8 ± 0.05
UniSRec (FHCM)	70.35 ± 0.04	79.73 ± 0.13	43.99 ± 0.12	46.37 ± 0.1	69.95 ± 0.06	79.84 ± 0.07	42.97 ± 0.18	45.48 ± 0.2
UniSRec	70.54 ± 0.09	79.74 ± 0.03	44.5 ± 0.06	46.84 ± 0.06	69.99 ± 0.07	79.63 ± 0.03	43.42 ± 0.08	45.87 ± 0.06
FDSA	68.94 ± 0.29	78.16 ± 0.13	48.62 ± 0.11	50.96 ± 0.08	67.88 ± 0.07	77.97 ± 0.11	47.04 ± 0.11	49.6 ± 0.12
S^3-Rec	62.82 ± 1.78	72.85 ± 1.01	40.84 ± 2.57	43.39 ± 2.37	60.6 ± 2.92	71.67 ± 2.19	37.88 ± 3.42	40.69 ± 3.22
LLM2SASRec	70.01 ± 0.1	79.15 ± 0.08	48.13 ± 0.08	50.45 ± 0.11	69.2 ± 0.14	79.11 ± 0.06	46.22 ± 0.39	48.73 ± 0.4
LLM2BERT4Rec	65.48 ± 0.02	75.91 ± 0.08	39.8 ± 0.16	42.45 ± 0.15	64.88 ± 0.44	75.9 ± 0.14	31.23 ± 0.26	32.0 ± 0.24
AlterRec	70.61 ± 0.03	79.75 ± 0.07	49.53 ± 0.02	**51.86 ± 0.01**	69.98 ± 0.01	79.75 ± 0.05	**47.87 ± 0.14**	**50.35 ± 0.14**
AlterRec_aug	**70.82 ± 0.09**	**79.84 ± 0.1**	**49.56 ± 0.06**	**51.86 ± 0.07**	**70.13 ± 0.03**	**79.86 ± 0.11**	**47.87 ± 0.13**	50.34 ± 0.15

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

[2]https://huggingface.co/sentence-transformers/distiluse-base-multilingual-cased-v1

[3]https://recbole.io/index.html

4.5.2 Performance Comparison

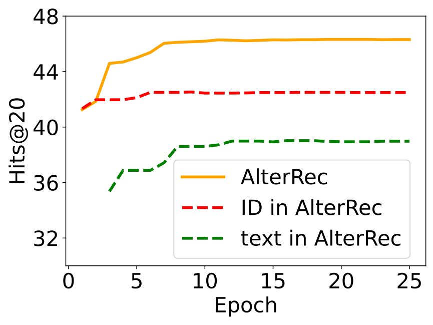
The comparison results are presented in Table 4.2. Since the Amazon-M2 dataset lacks user information, it is not feasible to obtain results for HG-GNN [81], which are denoted as “N/A”. Our observations are as follows: 1) Alter_aug consistently outperforms other baseline models across a range of datasets, with AlterRec often achieving the second-best performance, hightlighting the effectiveness of our alternative training strategy. Moreover, it demonstrates that integrating augmentation data can further enhance performance. Although UniSRec and FDSA exhibit strong performance in some cases, they do not consistently excel across all metrics. In contrast, AlterRec maintains a balanced and superior performance in both Hits@N and NDCG@N. For instance, AlterRec shows about a 10% relative improvement over UniSRec based on NDCG@10 and NDCG@20 on Amazon-M2 datasets. Additionally, it achieves approximately 19% and 2% relative improvements over FDSA based on Hits@10 and Hits@20 on the HD and Amazon-M2 datasets. 2) Models incorporating text data, like AlterRec, UniSRec, and FDSA, generally outperform ID-based models, indicating that text information offers complementary benefits and enhances overall performance.

Table 4.3 Ablation study on key components. Reported results (%) are mean value over three seeds.

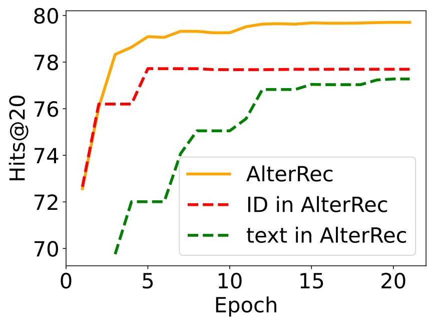
	HD		Amazon-French	
Methods	Hits@10	Hits@20	Hits@10	Hits@20
AlterRec	**38.25**	**46.31**	**70.61**	**79.75**
AlterRec_random	37.41	45.41	70.46	79.64
AlterRec_w/o_text	35.64	42.95	68.26	77.23
AlterRec_w/o_ID	30.05	38.73	66.96	76.85

4.5.3 Ablation Study

In this subsection, we evaluate the effectiveness of key components in our model: hard negative samples and the ID and text uni-modal networks. Table 4.3 presents the ablation study results for the following model variants: “AlterRec_random" for training with random negative samples, “AlterRec_w/o_text" for the model without the text uni-modal network, and “AlterRec_w/o_ID" for the model excluding the ID uni-modal network. AlterRec_w/o_text and AlterRec_w/o_ID are trained on a single modality.



(a) HD



(b) Amazon-French

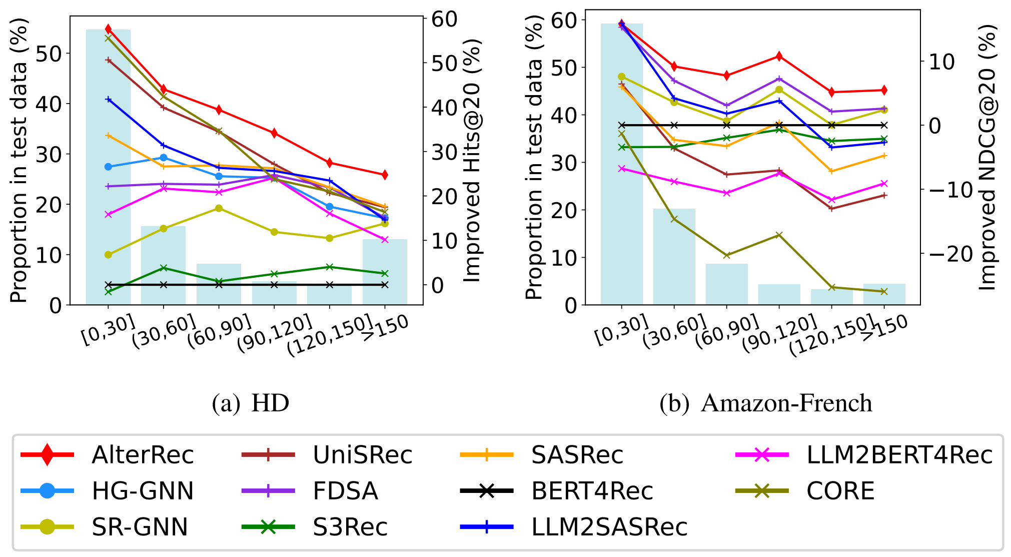
Figure 4.5 Test performance during the alternative training. AlterRec can mitigate the imbalance issue.

The results in Table 4.3 show that using random negative samples hurts performance, as it behave likes independent training and lacks interaction between the two modalities. This underscores the effectiveness of AlterRec over independent training, which benefits from hard negative samples to enhance learning between the two uni-modal networks. Furthermore, AlterRec significantly outperforms model variants that rely only on ID information, i.e., AlterRec_w/o_text. For instance, AlterRec achieves relative improvements of 7.82% and 3.26% in terms of Hits@20 on the HD and Amazon-French datasets, respectively. These findings highlight AlterRec's superior ability to integrate text information over naive fusion methods.

Additionally, we present the performance of AlterRec in Figure 4.5, including the individual performance of the ID and text components within AlterRec across epochs. These components are denoted as "ID in AlterRec" and "text in AlterRec", respectively. The overall performance of AlterRec is based on the score $y_{\mathbf{s},i}$ in Eq. (4.6). The performance of "ID in AlterRec" and "text in AlterRec" are derived from the scores $y_{\mathbf{s},i}^{ID}$ and $y_{\mathbf{s},i}^{text}$ within $y_{\mathbf{s},i}$. Figure 4.5 demonstrates that both ID and text components are effectively trained in our model, and crucially, AlterRec does not exhibit the imbalance issue commonly associated with naive fusion.

4.5.4 Performance on Long-tail Items

Textual data offers valuable semantic information that can be used to enhance long-tail items in session-based recommendation. To validate this, we divide the test data into groups based on the popularity of the ground-truth item in the training data. We then compare the performance of



(a) HD (b) Amazon-French

Figure 4.6 Performance comparison w.r.t. long-tail items. The bar graph depicts the proportion of sessions in the test data for each group. The line chart illustrates the improvement ratios for Hits@20 and NDCG@20 relative to BERT4Rec.

various methods in each group against the ID-based method BERT4Rec. The comparative result is presented in Figure 4.6, where we also show the proportion of each group. This figure reveals that a majority of items have sparse interactions (long-tail items). In most cases, AlterRec outperforms other baselines particularly on long-tail items.

For instance, AlterRec achieves the best performance in the [0,30] group on the HD and Amazon-French. It indicates that AlterRec effectively captures textual information, enhancing its performance on long-tail items.

Table 4.4 Comparison results (%) under the industrial setting. AlterRec achieves better performance than the deployed model.

Methods	HD-industry			
	Hits@10	Hits@20	NDCG@10	NDCG@20
Deployed_baseline	28.87	34.12	19.26	20.59
AlterRec	**29.74**	**35.31**	**19.68**	**21.09**
Improv.	3.01%	3.49%	2.18%	2.42%

4.5.5 Performance under the Industrial Setting

We further deploy our model in an industrial setting, training it on over 60 million sessions spanning the past two years and evaluating it on a validation set containing 0.6 million sessions from the following month. The training set includes over 1 million unique items, while the validation set contains more than 0.1 million items. We compare our model, AlterRec, with the previously deployed baseline model at The Home Depot, as reported in Table 4.4. The new dataset is referred to as "HD-Industry," the relative improvement is denoted as "Improv.," and the deployed baseline model is referred to as "Deployed_baseline." From the results, AlterRec demonstrates significantly better performance. For example, it achieves a 3.49% relative improvement over the baseline in terms of Hits@20. This highlights the effectiveness of AlterRec in industrial applications. In the following paragraph, we provide further details of model deployment.

Deployment. Our model has the same deployed setting with the baseline deployed model. AlterRec is deployed on Vertex AI [4], a fully managed machine learning platform on Google Cloud. The model is hosted and served via auto-generated APIs, enabling real-time inference and seamless integration with external applications. For inference, CPU instances are used to ensure a cost-effective and scalable deployment, maintaining low response times, while GPU instances are leveraged during training to accelerate model development. The model can be regularly retrained with fresh data using Google Cloud's AI Platform, with automated hyperparameter tuning incorporated to continuously optimize performance. Vertex AI's built-in tools can be used to track key metrics such as latency, accuracy, and conversion rates, with alerts set up for any anomalies. To handle cold-start items, when a new item is introduced to the system, an item representation is built online using item descriptions and titles—without requiring historical interaction data. This allows the system to generate recommendations for new items immediately. During periodic model retraining, ID-based embeddings can be constructed for items, providing a richer, more robust representation by learning from user-item interactions over time.

4.5.6 Parameter Analysis

In this subsection, we analyze the sensitivity of four key hyper-parameters: the parameter α which adjusts the contribution of ID and text scores in Eq. (4.6), β which is the weight assigned to the loss of augmented positive samples in Eq. (4.4) and Eq. (4.5); k_1 and k_2 which are

[4] `https://cloud.google.com/vertex-ai/?hl=en`

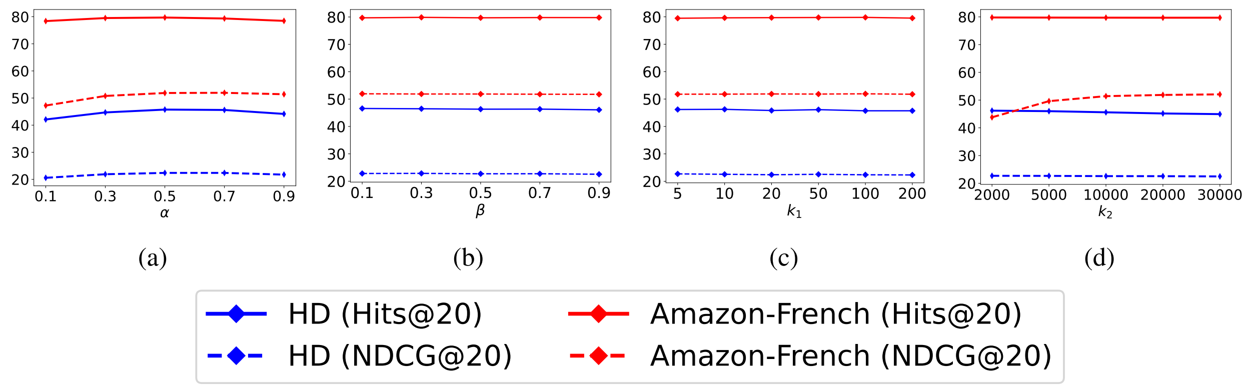


Figure 4.7 Performance of AlterRec by varying α, β, k_1 and k_2. AlterRec remains stable across a range of values.

the starting and end index used for selecting hard negative samples respectively, as discussed in Section 4.4.2. The results for Hits@20 and NDCG@20 on HD and Amazon-French are presented in Figure 4.7. Generally, AlterRec maintains stable performance across a range of hyper-parameter values, indicating that the model can achieve promising results with lightweight parameter tuning. Regarding α, an increase in performance is observed as α rises from 0.1 to 0.5, followed by a decrease when α is increased from 0.5 to 0.9. This suggests that an α value of 0.5 typically yields the best performance, indicating equal contributions from ID and text. For k_2, there is an increasing trend in NDCG@20 on Amazon-French and a decreasing trend in Hits@20 on HD as k_2 increases. This indicates that the Amazon-French may benefit from relatively more hard negatives, whereas HD does not require as many.

4.6 Conclusion

In this work, we explore an effective method for combining ID and text information in session-based recommendation. We have identified an imbalance issue in the widely-used naive fusion framework, which leads to insufficient integration of text information. To address this, we introduce a novel approach AlterRec employing the alternative training strategy that enables implicit interactions between ID and text, thereby facilitating their mutual learning and enhancing overall performance. Specifically, we develop separate uni-modal networks for ID and text to capture their respective information. By employing hard negative samples and augmented training samples from one network to train the other, we facilitate the exchange of information between the two, leading to improved overall performance. The effectiveness of AlterRec is validated through extensive experiments on various datasets against state-of-the-art baselines. In the future, we plan to investigate more advanced models, such as LLaMA, as the text encoders in AlterRec.

CHAPTER 5

LINK PREDICTION-ENHANCED LANGUAGE MODEL

Query understanding plays a key role in exploring users' search intents. However, it is inherently challenging since it needs to capture semantic information from short and ambiguous queries and often requires massive task-specific labeled data. In recent years, pre-trained language models (PLMs) have advanced various natural language processing tasks because they can extract general semantic information from large-scale corpora. However, directly applying them to query understanding is sub-optimal because existing strategies rarely consider to boost the search performance. On the other hand, search logs contain user clicks between queries and urls that provide rich users' search behavioral information on queries beyond their content. Therefore, in this paper, we aim to fill this gap by exploring search logs. In particular, we propose a novel graph-enhanced pre-training framework, GE-BERT, which leverages both query content and the query graph to capture both semantic information and users' search behavioral information of queries. Extensive experiments on offline and online tasks have demonstrated the effectiveness of the proposed framework.

5.1 Introduction

Query understanding [102] plays a crucial role in information retrieval. It aims to learn the intentions of a search query, and provides useful information to advance downstream applications such as document retrieval and ranking [44]. Many types of tasks have been developed to facilitate query understanding such as query classification [13, 99], query matching [56], and query expansion [152]. The challenges for these tasks are mainly from two aspects. First, search queries are often short and ambiguous; as a result, it is hard to capture the semantic meanings of queries only relying on their content. Second, the majority of existing solutions [13, 4] often train models from scratch in a supervised way, which require a large number of task-specific labeled data. However, obtaining such labeled data is labour intensive and usually costs tremendous money.

Recent years we have witnessed immense efforts in developing pre-trained language models (PLMs) [97, 43] such as BERT [22] and its variants [104, 136, 70]. These PLMs are trained on large-scale unlabeled corpora where they can extract general semantic information. Thus, they pave a way to help specific NLP tasks even when only a small amount of task-specific labeled data is available. However, directly applying these PLMs to query understanding can be sub-optimal. First, the goal of query understanding is to boost the search performance but the current PLMs seldom incorporate this goal into their pre-training strategies. Second, queries are different from normal natural languages because they are usually short.

To enhance the PLMs towards query understanding, one natural direction is to design domain-adaptive pre-training strategies with domain data [152, 78, 16, 69]. The search log is a commonly used domain data for query understanding, which is often denoted as a query-url bipartite click graph [44]. The graph encodes user search behaviors and provides rich information about queries beyond their content. Thus, it is appealing to explore this click graph to advance PLMs. However, BERT and its variants cannot directly process such graph data and dedicated efforts are desired.

In this paper, we propose a novel graph-based domain adaptive pre-training framework, named Graph Enhanced BERT(GE-BERT), which enjoys both advantages of the PLMs and the domain knowledge, i.e. click graph, for query understanding. To evaluate its effectiveness, we first fine-tune GE-BERT and compare it with representative baselines on two offline tasks, i.e., query classification and query clustering. Then we demonstrate the superiority of the fine-tuned GE-BERT on the online deployment based on the the medical query searching.

5.2 Related Work

In this section, we briefly reviews related work, including the query understanding tasks, general pre-training models, and various graph neural networks.

5.2.1 Query Understanding

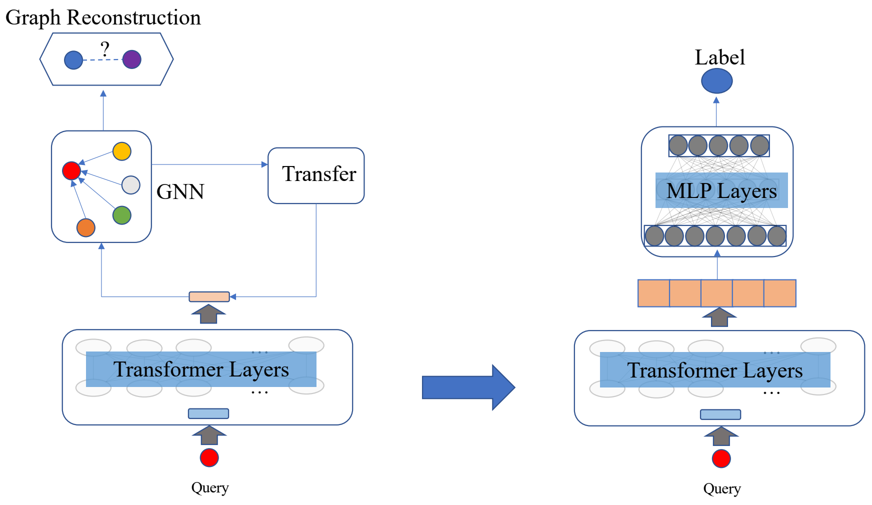
To explore users' query intents, researchers have conducted extensive studies from different perspectives, such as query classification [99, 154], query matching [56], query suggestion [122, 3, 75], query expansion [152], etc. Among them, query classification is an efective way to understand queries, which can straightforwardly improve the quality of Web search results. Query classification tries to categorize a query into a predefined taxonomy based on the topic [4, 13, 10, 132, 100]. Furthermore, query matching is another practical task for search engines, which aims to predict if two given queries have similar semantic meaning.

5.2.2 Pre-training Techniques

Strategies for learning pre-training language representations can be divided into two categories: feature-based and fine-tuning. Feature-based methods [77, 84, 85] generally adopt features to learn word embeddings. For example, ELMo [85] takes the pre-training representatios as additional features and generates high-quality deep context-aware emebddings, in which the word polysemy issue has been addressed. For fine-tuning based methods [86, 22], the goal is to provide a good pre-trained architecture that is used as a starting point for specific downstream tasks. The pre-trained model captures rich semantic patterns by training on large-scale corpus. Recent works [141, 88] have shown their effectiveness in various natural language processing areas. A notable example is BERT [22], which is built on the transformer architecture [111]. Recently, the effectiveness of BERT has also been explored in query understanding tasks [152, 78, 16, 69]. For example, Zheng et al. [152] proposed a query expansion model that leverages BERT to select relevant document chunks for expansion. Mustar et al. [78] showed that BERT exhibits more robust to noise and has a better understanding of complex queries than conventional methods for the query suggestion task.

5.2.3 Graph Neural Networks

Classical GNNs [12, 19, 54] are typically based on spectral methods, which generally define the convolutional operations in the Fourier domain. An efficient and simplified example is GCN [54], which extends the convolution operator by approximating the first-order Chebyshev expansion. Modern GNNs [34, 113] are built upon neural networks that learn representation vectors on the graph-structured data. They generally pass information from a target node to the next layer via propagation and aggregation on its neighbors. Among them, Velickovic et al. [113] proposed the GAT model that applies the attention mechanism to aggregate representations on the entire neighborhood. Inspired by the great success of GNNs, we adopt GNN to integrate the query graph information for accurately modeling.



(a) Pre-training (b) Fine-tuning

Figure 5.1 Illustration of pre-training and fine-tuning process. The Transformer module capture both the semantic and graph structural information after pretraining.

5.3 The Proposed Framework

In this section, we present the details of the proposed framework. Before that, we firstly introduce some key concepts and notations. The query graph is built from the query-url bipartite graph [44], where two queries are connected if they share at least one url in the bipartite graph. Formally, we denote the query graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, v_2, ..., v_n\}$ is the set of $|\mathcal{V}| = n$ query nodes, and $\mathcal{E}$ is the set of edges connecting nodes among $\mathcal{V}$. Furthermore, the query graph can also be described by an adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, where $\mathbf{A}_{ij} = 1$ if the nodes v_i and v_j are connected and $\mathbf{A}_{ij} = 0$ otherwise. Each query node v_i in the graph represents a query $q_i = \{c_1^i, \ldots, c_t^i\}$, which consists of a sequence of tokens. Here c_j^i indicates the j-th token in q_i, and t is the number of tokens.

Based on the query graph, we propose a pre-training framework GE-BERT to integrate the graph information into BERT, which can be further fine-tuned for downstream query understanding tasks. An illustration is shown in Figure 5.1. In particular, the pre-training framework in Figure 5.1(a) consists of three modules – the Transformer module, the GNN module and the Transfer module. The *graph semantics* captured from the GNN module is transferred through the Transfer module to the Transformer module. An illustration of the downstream task is presented in Figure 5.1(b).

Notably, we inject the query graph information into the Transformer module to perform downstream tasks, instead of using both the Transformer module and the GNN module to build the downstream model due to two reasons: 1) First, the GNN module is only trained on the existing query graph and thus it cannot handle new coming queries without any connection to the existing graph. The Transformer module, on the other hand, has the flexibility to generate embeddings for new queries. 2) Second, the complexity of the GNN module for fine-tuning and online deployment is significantly higher than that of the Transformer module, while the Transformer module provides more practical flexibility. Next, we introduce details of the three modules in the proposed pre-training framework and discuss pre-training strategies.

5.3.1 The Transformer Module

In the Transformer module, we adopt the Transformer encoder of the BERT model to learn representations of queries. The encoder is pre-trained on a large scale corpus and thus has strong ability to capture semantic information in natural language. The Transformer encoder takes a sequence of tokens as input and outputs a sequence of their corresponding final embeddings, i.e., one embedding for each token. To feed the queries into the Transformer encoder, we tokenize each query into a sequence of tokens based on the WordPiece method [127]. Following the original BERT [22], for each query, we add a special token [CLS] at the beginning of the token sequence; thus the query q_i is denoted as: $q_i = \{[CLS], c_1^i, \ldots, c_t^i\}$. We specifically denote the input embedding and output embedding for a token c_j^i in query q_i as $\mathbf{E}_{c_j^i}$ and $\mathbf{E}'_{c_j^i}$, respectively. Formally, we summarize the aforementioned process into the following formulation:

$$\{\mathbf{E}'_{CLS^i}, \mathbf{E}'_{c_1^i}, \ldots, \mathbf{E}'_{c_t^i}\} = \text{Transformer}(\{\mathbf{E}_{CLS^i}, \mathbf{E}_{c_1^i}, \ldots, \mathbf{E}_{c_t^i}\}). \tag{5.1}$$

Finally, we use the output embedding of the [CLS] token $\mathbf{E}'_{CLS^i}$ as the final embedding of the entire query. For convenience, we denote the final embedding output from the Transformer module for query q_i as $\mathbf{H}_i = \mathbf{E}'_{CLS^i}$.

5.3.2 The GNN Module

To capture the graph information in the query graph, we propose the GNN module which takes the query graph as input. The node's initial embedding is represented by the output embedding from the Transformer module for the specific query. The GNN module consists of several GNN layers, where each layer updates the node embedding by aggregating information from its neighborhood. Specifically, the GNN model is able to capture the K-hop neighborhood information after K layers. Formally, the k-th GNN layer can be generally defined as follows:

$$\tilde{\mathbf{H}}^k = \mathrm{GNN}(\mathbf{A}, \tilde{\mathbf{H}}^{k-1}) \tag{5.2}$$

where $\mathrm{GNN}(\cdot)$ denotes a general GNN layer, $\mathbf{A}$ is the adjacency matrix and $\tilde{\mathbf{H}}^k \in \mathbb{R}^{n \times d_k}$ is the output embedding of all nodes in the k-th layer. We use $\tilde{\mathbf{H}}_i \in \mathbb{R}^{d_K}$ to denote the final output embedding of the node v_i after a total of K iterations, i.e., $\tilde{\mathbf{H}}_i = \tilde{\mathbf{H}}_i^K$. Specifically, the node embedding is initialized with the output from the Transformer module, i.e., $\tilde{\mathbf{H}}_i^0 = \mathbf{H}_i$.

In this paper, we adopt two representative GNN models, i.e., GCN [54] and GAT [113], to capture the query graph information. To preserve the graph information and train the GNN parameters, we use the output node embeddings from the GNN module to reconstruct the query graph. In particular, given two nodes v_i and v_j, we define a probability score $s_{ij} = \mathrm{sigmoid}(\tilde{\mathbf{H}}_i \cdot \tilde{\mathbf{H}}_j)$ to predict if they have a link in the query graph where $\mathrm{sigmoid}(\cdot)$ is the sigmoid function. The loss of the graph reconstruction is defined as follows:

$$\mathcal{L}_{GNN} = -\sum_i \sum_{j \in (\mathcal{N}_i \cup \mathcal{N}_i')} \mathbf{A}_{ij} \log s_{ij} + (1 - \mathbf{A}_{ij}) \log(1 - s_{ij}) \tag{5.3}$$

where $\mathcal{N}_i$ is the set of neighbors of v_i, $\mathcal{N}_i'$ is the set of negative samples that are not connected with node v_i, and $|\mathcal{N}_i'| = |\mathcal{N}_i|$.

5.3.3 The Transfer Module

The Transfer module is proposed to fuse the graph information captured by the GNN module to the Transformer module. In the literature, minimizing the KL divergence between prior and posterior distributions is demonstrated to be an effective way for knowledge transfer [144, 65]. Inspired by this, we define the prior distribution based on the Transformer module since it only utilizes query content: $p(\mathbf{A}_{ij}|\mathbf{H}_i, \mathbf{H}_j) = sigmoid(\mathbf{H}_i \cdot \mathbf{H}_j)$. The posterior distribution is obtained from the GNN module as it involves additional information from users' search behaviors: $\tilde{p}(\mathbf{A}_{ij}|\tilde{\mathbf{H}}_i, \tilde{\mathbf{H}}_j) = sigmoid(\tilde{\mathbf{H}}_i \cdot \tilde{\mathbf{H}}_j)$. The loss to minimize the distance between these two distributions is defined as follows:

$$\mathcal{L}_{KL} = \sum_i \sum_{j \in (\mathcal{N}_i \cup \mathcal{N}'_i)} \Bigg(\tilde{p}(\mathbf{A}_{ij}|\tilde{\mathbf{H}}_i, \tilde{\mathbf{H}}_j) \log \frac{\tilde{p}(\mathbf{A}_{ij}|\tilde{\mathbf{H}}_i, \tilde{\mathbf{H}}_j)}{p(\mathbf{A}_{ij}|\mathbf{H}_i, \mathbf{H}_j)} + (1 - \tilde{p}(\mathbf{A}_{ij}|\tilde{\mathbf{H}}_i, \tilde{\mathbf{H}}_j)) \log \frac{1 - \tilde{p}(\mathbf{A}_{ij}|\tilde{\mathbf{H}}_i, \tilde{\mathbf{H}}_j)}{1 - p(\mathbf{A}_{ij}|\mathbf{H}_i, \mathbf{H}_j)} \Bigg)$$

The prior and posterior distributions essentially predict if two nodes are connected. Optimizing this loss potentially drives the Transformer module to achieve similar graph reconstruction result with the GNN module, i.e., the Transformer module could naturally capture the graph information guided by the GNN module.

5.3.4 Pre-training Strategies

We propose two strategies for the pre-training process, i.e., stage-by-stage and joint training. 1) In the **stage-by-stage strategy**, we separate the training processes of the Transformer module and the GNN module. In the stage of training the GNN module, we fix the parameters of the Transformer module and update the GNN module by the loss $\mathcal{L}_{GNN}$. In the stage of training the Transformer Module, we fix the GNN module and transfer the information captured by the GNN module into the Transformer module by the loss $\mathcal{L}_{KL}$. 2) In the **joint training strategy**, the proposed modules are trained simultaneously. The loss function is defined by combining two losses: $\mathcal{L}_J = \mathcal{L}_{GNN} + \lambda \mathcal{L}_{KL}$, where λ is a pre-defined parameter to balance the two losses. For both strategies, the trained Transformer module can be used to build models for downstream tasks.

5.4 Experiment

In this section, we aim to validate the effectiveness of GE-BERT. We first fine-tune it on two **offline** query understanding tasks, i.e., query classification and query matching tasks. Then, we further demonstrate the effectiveness of the fine-tuned model in the **online** search system based on the medical query searching.

5.4.1 Datasets

We use different datasets according to the tasks. 1) For the **pre-training**, the model is trained on the query graph which is built on a large query log data generated by the Baidu search engine[1] with millions of users. We collected the initial search log data within a week to construct the query graph which consists of $92,413,932$ query nodes and $957,210,956$ edges. 2) In the **offline query classification** task, we predict the label for the input query. Each query is associated with a label such as music and medical care. There are $100,887$ queries with 30 classes in total. 3) In the **offline query matching** task, we predict if the input query pairs have similar semantic meaning or not. Each query pair is associated with a label which is 1 when the two queries have similar semantic meanings, and 0 otherwise. There are $108,615$ query pairs with 2 classes in total. We adopt $60\%, 20\%, 20\%$ for the training/validation/test split for offline tasks.

5.4.2 Offline tasks

5.4.2.1 Baselines and Model Variants

We consider two baselines based on the BERT model [22]. The first one directly uses BERT model in the downstream task, denoted as **BERT**. The second one further trains on the query graph under the graph reconstruction task, and we denote it as **BERT+Q**. Namely, different from the **BERT**, **BERT+Q** further trains the BERT parameters on the query graph by the loss function which has the same form as Eq.(5.3). **BERT+Q** is a baseline to evaluate the effectiveness of the GNN module to capture query graph information. Two variants are **GE-BERT-J** which jointly trains all proposed modules and **GE-BERT-S** which uses the stage-by-stage pre-training strategy.

5.4.2.2 Experimental Settings

For pre-training tasks, the number of stages in GE-BERT-S is set to be 4, and λ in GE-BERT-J is set to be 1. For the two downstream tasks, we utilize the Transformer module to build models. Specifically, for the query classification task, given a query, we feed the query embedding from the Transformer Module into a Multi-Layer Perceptron (MLP) for classification. For the query matching task, given a query pair, we feed their embeddings into a MLP and then calculate the inner product

[1]https://www.baidu.com/

Table 5.1 Performance of the offline query classification task. The split of the dataset is 60%, 20%, 20% for training,validation and test, respectively.

Methods		ACC	Precision	Recall	F1
BERT		0.6728	0.5941	0.5604	0.5722
BERT+Q		0.6862	0.6162	0.5708	0.5863
GAT-based	GE-BERT-J	0.6897	0.6215	0.5702	0.5874
	GE-BERT-S	0.7033	**0.6308**	0.5835	0.5997
GCN-based	GE-BERT-J	0.6945	0.6292	0.5758	0.5923
	GE-BERT-S	**0.7034**	0.6227	**0.5978**	**0.6034**

Table 5.2 Performance of the offline query matching task. The split of the dataset is 60%, 20%, 20% for training,validation and test, respectively.

Methods		ACC	Precision	Recall	F1
BERT		0.6232	0.7825	0.6282	0.5661
BERT+Q		0.6254	0.7842	0.6303	0.5691
GAT-based	GE-BERT-J	0.6388	0.7887	0.6436	0.5891
	GE-BERT-S	0.6370	0.7881	0.6418	0.5865
GCN-based	GE-BERT-J	**0.6425**	**0.7899**	**0.6472**	**0.5944**
	GE-BERT-S	0.6312	0.7861	0.6317	0.5779

followed by the sigmoid function to generate the score for final prediction.

5.4.2.3 Performance

The experiment results are shown in Table 5.1 and Table 5.2 respectively. From these tables, we can make the following observations: 1) BERT+Q outperforms BERT consistently which demonstrates that incorporating query graph information can enhance Transformer to perform better. 2) In general, the proposed GE-BERT variants achieve better performance than BERT+Q, which indicates that the GNN module can capture the query graph information more effectively than pre-training with simple graph reconstruction task, which facilitates the performance. It also suggests that KL-divergence can indeed facilitate the process of transfer the information from the GNN module to the Transformer module.

5.4.3 Online task

In this subsection, we conduct an online A/B testing to validate the effectiveness of our models in the Baidu search engine. It is based on the final ranked results for medical query searching, which is an important search scenario in the Baidu search engine. A crucial step for the medical query searching is to identify medical queries first. To achieve this goal, we treat it as a classification task and directly utilize the **fine-tuned** model (including the MLP layer) from the offline query classification task to perform the classification. In particular, given an input query, the fine-tuned model is able to tell whether it is a medical query or not. We have designed many strategies for medical queries to facilitate their ranking performance. Thus, different fine-tuned models might have different classification performance, which leads to different final ranked results on the search engine. We evaluate the model performance by comparing their ranked results. Due to the high cost of deploying multiple models, we evaluate the fine-tuned GE-BERT-S with GCN (**GE-BERT-S+F**) and make a comparison with the fine-tuned BERT+Q model (**BERT+Q+F**)

5.4.3.1 Experimental Settings and Result

We randomly sample a certain number of queries from the search log, and we apply GE-BERT-S+F and BERT+Q+F to identify the medical queries. Then we obtain $6,268$ queries which are predicted as medical queries by as least one of the two models. Among these queries, they make different predictions on 964 queries, i.e., the difference ratio is $964/6268 = 15.38\%$.

To compare the performance of GE-BERT-S+F and BERT+Q+F online, we compare their ranked results obtained from the search engine for these 964 queries, and use the (Good vs. Same vs. Bad) GSB [159] as the metric. More specifically, given a query, the annotators are provided with a pair $(result_1, result_2)$ where $result_1$ is the ranked result from GE-BERT-S+F and $result_2$ is the ranked result from BERT+Q+F. The result consists of multiple URLs. The annotators who don't know which model the result is generated are asked to rate the result independently: Good ($result_1$ is better), Bad ($result_2$ is better), and Same (they are equally good or bad) by considering the relevance between the ranked result and the given query. To quantify the human evaluation, the ΔGSB is defined by combining these three indicators:

$$\Delta GSB = \frac{\text{\#Good} - \text{\#Bad}}{\text{\#Good} + \text{\#Same} + \text{\#Bad}} \tag{5.4}$$

The statistic for computing the ΔGSB is shown in Table 5.3. We can observe that GE-BERT-S+F brings in substantial improvement and the advantage of GSB is 3.20%, which shows the superiority of our model to advance the online medical query searching.

Table 5.3 Results of the online A/B testing. Our fine-tuned model achieves better results than the baseline model.

diff_ratio	# Good	# Same	# Bad	ΔGSB
15.38%	57	881	26	3.20%

Table 5.4 Case study of query classification for the online task. Our fine-tuned model achieves better results than the baseline model.

Query	Label	GE-BERT-S+F	BERT+Q+F
What's the weight of Nini	people	people	medical care
Dead Cells the fisherman	games	games	medical care
Which eyedrops is good for long-term use for students	medical care	medical care	product
What are the pros and cons of drinking soy milk	medical care	medical care	parenting

5.4.3.2 Case Study

In this section, we further analyze the fine-tuned model by practical cases in query classification. They are presented in Table 5.4 where the second column is the true label of the query, the third and the forth column are predicted labels from GE-BERT-S+F and BERT+Q+F respectively. We have the following observations: 1) For the first two queries, BERT+Q+F wrongly predicts them as medical queries. The potential reason might come from two misleading words *weight* and *Cells* because they are related to medical care. However, the first query is more about the word *Nini* (a famous actress in China) and *Dead Cells* in the second query is a video game. 2) For the third and the forth query, BERT+Q+F wrongly predicts them as other categories instead of the medical query. However, the word *eyedrops* in the third query is not simply a product but more related to the medical care. In the forth query, the word *milk* might be misleading so BERT+Q+F predicts it as *parenting*. In fact, the user cares more about the function of the soy milk so it's a medical query. Generally, these queries are difficult ones containing the misleading words. GE-BERT-S+F is able to make the right predictions. The potential reason is that the GNN utilizes the information of neighboring queries which can help to distinguish the misleading words.

5.5 Conclusion

In this paper, we introduce GE-BERT, a novel pre-training framework preserving both the semantic information and query graph information, which is further fine-tuned in the query understanding tasks. Specifically, we devise three modules: a Transformer module for extracting the semantic information of query content, a GNN module for capturing the query graph information, and a Transfer module to transfer the graph information from the GNN module to the Transformer module. To train the GE-BERT, two strategies are proposed: stage-by-stage strategy by separating the training of the Transformer module and the GNN module, and joint training by training all modules simultaneously. Comprehensive experiments in offline and online tasks demonstrate the superiority of GE-BERT.

CHAPTER 6

CONCLUSION AND FUTURE DIRECTIONS

In this chapter, we summarize the research results of this dissertation, discuss their broader impact and highlight promising research direction.

6.1 Summary

In this dissertation, we study to address critical challenges in link prediction, a foundational task in graph data analysis with extensive real-world applications. By identifying and addressing evaluation pitfalls across diverse graph types, this work seeks to establish a more rigorous and unified framework for assessing link prediction methods. Additionally, by leveraging the capabilities of language models to integrate textual attributes with graph data, we try to enhance the semantic understanding and practical performance of link prediction in various domains. Ultimately, these contributions will not only advance the field of graph machine learning but also foster the development of more robust and impactful applications in areas such as social networks, recommendation systems, and beyond.

6.2 Future Work

In the future, we plan to further explore from the following perspectives:

- I am interested in integrating graphs with large language models (LLMs) [60] to drive advancements in AI applications. Compared to the smaller language models used in my previous research, LLMs are significantly larger and more powerful, though they also present new challenges. For instance, LLMs may suffer from hallucination issues, as they rely on learned patterns without true factual understanding. A promising solution involves using graph structures for retrieval, which can help LLMs capture more reliable, contextually relevant information and improve multi-hop reasoning. LLMs also offer substantial benefits for graph-based tasks. Their proficiency with text data can enhance embedding representations, feature enrichment, feature generation, and graph construction.
- I aim to leverage my expertise in AI and ML to make a positive societal impact through interdisciplinary research that drives innovation and contributes to social good. I'm interested in exploring AI4Science, applying AI and ML to accelerate scientific discovery across disciplines such

as biology, medicine, and environmental science. By integrating multi-modal data, such as text and images, and employing advanced multi-modal models, I aim to advance various interdisciplinary applications, contributing to innovations in areas such as healthcare, environmental science, and education.

BIBLIOGRAPHY

[1] Lada A Adamic and Eytan Adar. Friends and neighbors on the web. Social networks, 25(3):211–230, 2003.

[2] Mehdi Ali, Max Berrendorf, Charles Tapley Hoyt, Laurent Vermue, Mikhail Galkin, Sahand Sharifzadeh, Asja Fischer, Volker Tresp, and Jens Lehmann. Bringing light into the dark: A large-scale evaluation of knowledge graph embedding models under a unified framework. IEEE Transactions on Pattern Analysis and Machine Intelligence, 44(12):8825–8845, 2021.

[3] Aris Anagnostopoulos, Luca Becchetti, Carlos Castillo, and Aristides Gionis. An optimization framework for query recommendation. In Proceedings of the third ACM international conference on Web search and data mining, pages 161–170, 2010.

[4] Steven M Beitzel, Eric C Jensen, Ophir Frieder, David D Lewis, Abdur Chowdhury, and Aleksander Kolcz. Improving automatic query classification via semi-supervised learning. In Fifth IEEE International Conference on Data Mining (ICDM'05), pages 8–pp. IEEE, 2005.

[5] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pages 1247–1250, 2008.

[6] JC de Borda. M'emoire sur les' elections au scrutin. Histoire de l'Acad'emie Royale des Sciences, 1781.

[7] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. Advances in neural information processing systems, 26, 2013.

[8] Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States, pages 2787–2795, 2013.

[9] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. Computer networks and ISDN systems, 30(1-7):107–117, 1998.

[10] Andrei Z Broder, Marcus Fontoura, Evgeniy Gabrilovich, Amruta Joshi, Vanja Josifovski, and Tong Zhang. Robust classification of rare queries using web knowledge. In Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, pages 231–238, 2007.

[11] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. Advances in neural information processing systems, 33:1877–1901, 2020.

[12] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. In 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings, 2014.

[13] Huanhuan Cao, Derek Hao Hu, Dou Shen, Daxin Jiang, Jian-Tao Sun, Enhong Chen, and Qiang Yang. Context-aware query classification. In Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval, pages 3–10, 2009.

[14] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R Hruschka, and Tom M Mitchell. Toward an architecture for never-ending language learning. In Twenty-Fourth AAAI conference on artificial intelligence, 2010.

[15] Benjamin Paul Chamberlain, Sergey Shirobokov, Emanuele Rossi, Fabrizio Frasca, Thomas Markovich, Nils Hammerla, Michael M Bronstein, and Max Hansmire. Graph neural networks for link prediction with subgraph sketching. arXiv preprint arXiv:2209.15486, 2022.

[16] Qian Chen, Zhu Zhuo, and Wen Wang. Bert for joint intent classification and slot filling. arXiv preprint arXiv:1902.10909, 2019.

[17] Nur Nasuha Daud, Siti Hafizah Ab Hamid, Muntadher Saadoon, Firdaus Sahran, and Nor Badrul Anuar. Applications of link prediction in social networks: A review. Journal of Network and Computer Applications, 166:102716, 2020.

[18] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In Proceedings of the 23rd international conference on Machine learning, pages 233–240, 2006.

[19] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain, pages 3837–3845, 2016.

[20] Walter H Delashmit, Michael T Manry, et al. Recent developments in multilayer perceptron neural networks. In Proceedings of the seventh annual memphis area engineering and science conference, MAESC, pages 1–15, 2005.

[21] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018, pages 1811–1818. AAAI Press, 2018.

[22] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers), pages 4171–4186, 2019.

[23] Chenzhuang Du, Jiaye Teng, Tingle Li, Yichen Liu, Tianyuan Yuan, Yue Wang, Yang Yuan, and Hang Zhao. On uni-modal feature learning in supervised multi-modal learning. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA, volume 202 of Proceedings of Machine Learning Research, pages 8632–8656. PMLR, 2023.

[24] Chenzhuang Du, Jiaye Teng, Tingle Li, Yichen Liu, Tianyuan Yuan, Yue Wang, Yang Yuan, and Hang Zhao. On uni-modal feature learning in supervised multi-modal learning. arXiv preprint arXiv:2305.01233, 2023.

[25] Vijay Prakash Dwivedi, Chaitanya K Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. Journal of Machine Learning Research, 24(43):1–48, 2023.

[26] Chantat Eksombatchai, Pranav Jindal, Jerry Zitao Liu, Yuchen Liu, Rahul Sharma, Charles Sugnet, Mark Ulrich, and Jure Leskovec. Pixie: A system for recommending 3+ billion items to 200+ million users in real-time. In Proceedings of the 2018 world wide web conference, pages 1775–1784, 2018.

[27] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In Proceedings of the 8th International Conference on Learning Representations (ICLR), 2020.

[28] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In The World Wide Web Conference, pages 417–426, 2019.

[29] Christiane Fellbaum. Wordnet. In Theory and applications of ontology: computer applications, pages 231–243. Springer, 2010.

[30] Alberto García-Durán, Sebastijan Dumančić, and Mathias Niepert. Learning sequence encoders for temporal knowledge graph completion. arXiv preprint arXiv:1809.03202, 2018.

[31] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, pages 855–864, 2016.

[32] Pankaj Gupta, Ashish Goel, Jimmy Lin, Aneesh Sharma, Dong Wang, and Reza Zadeh. Wtf: The who to follow service at twitter. In Proceedings of the 22nd international conference on World Wide Web, pages 505–514, 2013.

[33] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. Advances in neural information processing systems, 30, 2017.

[34] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, pages 1024–1034, 2017.

[35] Jesse Harte, Wouter Zorgdrager, Panos Louridas, Asterios Katsifodimos, Dietmar Jannach, and Marios Fragkoulis. Leveraging large language models for sequential recommendation. In Proceedings of the 17th ACM Conference on Recommender Systems, pages 1096–1102, 2023.

[36] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. In Yoshua Bengio and Yann LeCun, editors, 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings, 2016.

[37] Yupeng Hou, Binbin Hu, Zhiqiang Zhang, and Wayne Xin Zhao. Core: simple and effective session-based recommendation within consistent representation space. In Proceedings of the 45th international ACM SIGIR conference on research and development in information retrieval, pages 1796–1801, 2022.

[38] Yupeng Hou, Shanlei Mu, Wayne Xin Zhao, Yaliang Li, Bolin Ding, and Ji-Rong Wen. Towards universal sequence representation learning for recommender systems. In Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pages 585–593, 2022.

[39] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. Advances in neural information processing systems, 33:22118–22133, 2020.

[40] Yu Huang, Junyang Lin, Chang Zhou, Hongxia Yang, and Longbo Huang. Modality competition: What makes joint training of multi-modal network fail in deep learning?(provably). In International Conference on Machine Learning, pages 9226–9259. PMLR, 2022.

[41] Zan Huang, Xin Li, and Hsinchun Chen. Link prediction approach to collaborative filtering. In Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries, pages 141–142, 2005.

[42] Zexi Huang, Mert Kosan, Arlei Silva, and Ambuj Singh. Link prediction without graph neural networks. arXiv preprint arXiv:2305.13656, 2023.

[43] Chanwoo Jeong, Sion Jang, Eunjeong Park, and Sungchul Choi. A context-aware citation recommendation model with bert and graph convolutional networks. Scientometrics, 124:1907–1922, 2020.

[44] Shan Jiang, Yuening Hu, Changsung Kang, Tim Daly Jr, Dawei Yin, Yi Chang, and Chengxiang Zhai. Learning query and document relevance from a web-scale click graph. In Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval, pages 185–194, 2016.

[45] Wei Jin, Haitao Mao, Zheng Li, Haoming Jiang, Chen Luo, Hongzhi Wen, Haoyu Han, Hanqing Lu, Zhengyang Wang, Ruirui Li, et al. Amazon-m2: A multilingual multi-locale shopping session dataset for recommendation and text generation. arXiv preprint arXiv:2307.09688, 2023.

[46] Wei Jin, Lingxiao Zhao, Shichang Zhang, Yozen Liu, Jiliang Tang, and Neil Shah. Graph condensation for graph neural networks. arXiv preprint arXiv:2110.07580, 2021.

[47] Woojeong Jin, Meng Qu, Xisen Jin, and Xiang Ren. Recurrent event network: Autoregressive structure inference over temporal knowledge graphs. arXiv preprint arXiv:1904.05530, 2019.

[48] Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In 2018 IEEE international conference on data mining (ICDM), pages 197–206. IEEE, 2018.

[49] Leo Katz. A new status index derived from sociometric analysis. Psychometrika, 18(1):39–43, 1953.

[50] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of naacL-HLT, volume 1, page 2, 2019.

[51] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[52] Thomas N Kipf and Max Welling. Variational graph auto-encoders. arXiv preprint arXiv:1611.07308, 2016.

[53] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In International Conference on Learning Representations (ICLR), 2017.

[54] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, 2017.

[55] István A Kovács, Katja Luck, Kerstin Spirohn, Yang Wang, Carl Pollis, Sadie Schlabach, Wenting Bian, Dae-Kyum Kim, Nishka Kishore, Tong Hao, et al. Network-based prediction of protein interactions. Nature communications, 10(1):1240, 2019.

[56] Hang Li and Jun Xu. Semantic matching in search. Foundations and Trends in Information retrieval, 7(5):343–469, 2014.

[57] Jian Li, Jieming Zhu, Qiwei Bi, Guohao Cai, Lifeng Shang, Zhenhua Dong, Xin Jiang, and Qun Liu. Miner: multi-interest matching network for news recommendation. In Findings of the Association for Computational Linguistics: ACL 2022, pages 343–352, 2022.

[58] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. Neural attentive session-based recommendation. In Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, pages 1419–1428, 2017.

[59] Juanhui Li, Haoyu Han, Zhikai Chen, Harry Shomer, Wei Jin, Amin Javari, and Jiliang Tang. Enhancing id and text fusion via alternative training in session-based recommendation. arXiv preprint arXiv:2402.08921, 2024.

[60] Juanhui Li, Sreyashi Nag, Hui Liu, Xianfeng Tang, Sheikh Sarwar, Limeng Cui, Hansu Gu, Suhang Wang, Qi He, and Jiliang Tang. Learning with less: Knowledge distillation from large language models via unlabeled data. arXiv preprint arXiv:2411.08028, 2024.

[61] Juanhui Li, Harry Shomer, Jiayuan Ding, Yiqi Wang, Yao Ma, Neil Shah, Jiliang Tang, and Dawei Yin. Are message passing neural networks really helpful for knowledge graph completion? In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 10696–10711, Toronto, Canada, July 2023. Association for Computational Linguistics.

[62] Juanhui Li, Harry Shomer, Haitao Mao, Shenglai Zeng, Yao Ma, Neil Shah, Jiliang Tang, and Dawei Yin. Evaluating graph neural networks for link prediction: Current pitfalls and new benchmarking. Advances in Neural Information Processing Systems, 36, 2024.

[63] Juanhui Li, Wei Zeng, Suqi Cheng, Yao Ma, Jiliang Tang, Shuaiqiang Wang, and Dawei Yin. Graph enhanced bert for query understanding. In Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 3315–3319, 2023.

[64] Ruyu Li, Wenhao Deng, Yu Cheng, Zheng Yuan, Jiaqi Zhang, and Fajie Yuan. Exploring the upper limits of text-based collaborative filtering using large language models: Discoveries and insights. arXiv preprint arXiv:2305.11700, 2023.

[65] Rongzhong Lian, Min Xie, Fan Wang, Jinhua Peng, and Hua Wu. Learning to select knowledge for response generation in dialog systems. arXiv preprint arXiv:1902.04911, 2019.

[66] David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. In Proceedings of the twelfth international conference on Information and knowledge management, pages 556–559, 2003.

[67] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In Twenty-ninth AAAI conference on artificial intelligence, 2015.

[68] Hanxiao Liu, Zihang Dai, David R So, and Quoc V Le. Pay attention to mlps. arXiv preprint arXiv:2105.08050, 2021.

[69] Xiaowei Liu, Weiwei Guo, Huiji Gao, and Bo Long. Deep search query intent understanding. CoRR, abs/2008.06759, 2020.

[70] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692, 2019.

[71] Linyuan Lü and Tao Zhou. Link prediction in complex networks: A survey. Physica A: statistical mechanics and its applications, 390(6):1150–1170, 2011.

[72] Sichun Luo, Chen Ma, Yuanzhang Xiao, and Linqi Song. Improving long-tail item recommendation with graph augmentation. In Proceedings of the 32nd ACM international conference on information and knowledge management, pages 1707–1716, 2023.

[73] Yao Ma, Xiaorui Liu, Tong Zhao, Yozen Liu, Jiliang Tang, and Neil Shah. A unified view on graph neural networks as graph signal denoising. In Proceedings of the 30th ACM International Conference on Information & Knowledge Management, pages 1202–1211, 2021.

[74] Haitao Mao, Juanhui Li, Harry Shomer, Bingheng Li, Wenqi Fan, Yao Ma, Tong Zhao, Neil Shah, and Jiliang Tang. Revisiting link prediction: A data perspective. arXiv preprint arXiv:2310.00793, 2023.

[75] Qiaozhu Mei, Dengyong Zhou, and Kenneth Church. Query suggestion using hitting time. In Proceedings of the 17th ACM conference on Information and knowledge management, pages 469–478, 2008.

[76] Aditya Krishna Menon and Charles Elkan. Link prediction via matrix factorization. In Joint european conference on machine learning and knowledge discovery in databases, pages 437–452. Springer, 2011.

[77] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. arXiv preprint arXiv:1310.4546, 2013.

[78] Agnès Mustar, Sylvain Lamprier, and Benjamin Piwowarski. Using bert and bart for query suggestion. In Joint Conference of the Information Retrieval Communities in Europe, volume 2621. CEUR-WS. org, 2020.

[79] Deepak Nathani, Jatin Chauhan, Charu Sharma, and Manohar Kaul. Learning attention-based embeddings for relation prediction in knowledge graphs. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 4710–4723, 2019.

[80] Mark EJ Newman. Clustering and preferential attachment in growing networks. Physical review E, 64(2):025102, 2001.

[81] Yitong Pang, Lingfei Wu, Qi Shen, Yiming Zhang, Zhihua Wei, Fangli Xu, Ethan Chang, Bo Long, and Jian Pei. Heterogeneous global graph neural networks for personalized session-based recommendation. In Proceedings of the fifteenth ACM international conference on web search and data mining, pages 775–783, 2022.

[82] Yoon-Joo Park and Alexander Tuzhilin. The long tail of recommender systems and how to leverage it. In Proceedings of the 2008 ACM conference on Recommender systems, pages 11–18, 2008.

[83] Xiaokang Peng, Yake Wei, Andong Deng, Dong Wang, and Di Hu. Balanced multimodal learning via on-the-fly gradient modulation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 8238–8247, 2022.

[84] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pages 1532–1543, 2014.

[85] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. arXiv preprint arXiv:1802.05365, 2018.

[86] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

[87] Saeed Rahmani, Asiye Baghbani, Nizar Bouguila, and Zachary Patterson. Graph neural networks for intelligent transportation systems: A survey. IEEE Transactions on Intelligent Transportation Systems, 24(8):8846–8885, 2023.

[88] Prajit Ramachandran, Peter J Liu, and Quoc V Le. Unsupervised pretraining for sequence to sequence learning. arXiv preprint arXiv:1611.02683, 2016.

[89] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. arXiv preprint arXiv:1908.10084, 2019.

[90] Xubin Ren, Wei Wei, Lianghao Xia, Lixin Su, Suqi Cheng, Junfeng Wang, Dawei Yin, and Chao Huang. Representation learning with large language models for recommendation. arXiv preprint arXiv:2310.15950, 2023.

[91] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, pages 452–461, 2009.

[92] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. arXiv preprint arXiv:1205.2618, 2012.

[93] Daniel Ruffinelli, Samuel Broscheit, and Rainer Gemulla. You can teach an old dog new tricks! on training knowledge graph embeddings. In International Conference on Learning Representations.

[94] Takaya Saito and Marc Rehmsmeier. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. PloS one, 10(3):e0118432, 2015.

[95] Aravind Sankar, Yozen Liu, Jun Yu, and Neil Shah. Graph neural networks for friend ranking in large-scale social platforms. In Proceedings of the Web Conference 2021, pages 2535–2546, 2021.

[96] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In European semantic web conference, pages 593–607. Springer, 2018.

[97] Junyuan Shang, Tengfei Ma, Cao Xiao, and Jimeng Sun. Pre-training of graph augmented transformers for medication recommendation. arXiv preprint arXiv:1906.00346, 2019.

[98] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. arXiv preprint arXiv:1811.05868, 2018.

[99] Dou Shen, Rong Pan, Jian-Tao Sun, Jeffrey Junfeng Pan, Kangheng Wu, Jie Yin, and Qiang Yang. Query enrichment for web-query classification. ACM Transactions on Information Systems (TOIS), 24(3):320–352, 2006.

[100] Yangyang Shi, Kaisheng Yao, Le Tian, and Daxin Jiang. Deep lstm based feature mapping for query classification. In Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: Human language technologies, pages 1501–1511, 2016.

[101] Harry Shomer, Yao Ma, Haitao Mao, Juanhui Li, Bo Wu, and Jiliang Tang. Lpformer: An adaptive graph transformer for link prediction. In Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pages 2686–2698, 2024.

[102] Giorgos Stoilos, Nikos Papasarantopoulos, Pavlos Vougiouklis, and Patrik Bansky. Type linking for query understanding and semantic search. In Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pages 3931–3940, 2022.

[103] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In Proceedings of the 28th ACM international conference on information and knowledge management, pages 1441–1450, 2019.

[104] Yu Sun, Shuohuan Wang, Yukun Li, Shikun Feng, Xuyi Chen, Han Zhang, Xin Tian, Danxiang Zhu, Hao Tian, and Hua Wu. Ernie: Enhanced representation through knowledge integration. arXiv preprint arXiv:1904.09223, 2019.

[105] Zhiqing Sun, Shikhar Vashishth, Soumya Sanyal, Partha Talukdar, and Yiming Yang. A re-evaluation of knowledge graph completion methods. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 5516–5522, 2020.

[106] Hamidreza Tahmasbi, Mehrdad Jalali, and Hassan Shakeri. Modeling user preference dynamics with coupled tensor factorization for social media recommendation. Journal of Ambient Intelligence and Humanized Computing, 12:9693–9712, 2021.

[107] Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, et al. Mlp-mixer: An all-mlp architecture for vision. arXiv preprint arXiv:2105.01601, 2021.

[108] Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In Proceedings of the 3rd workshop on continuous vector space models and their compositionality, pages 57–66, 2015.

[109] Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. Representing text for joint embedding of text and knowledge bases. In Proceedings of the 2015 conference on empirical methods in natural language processing, pages 1499–1509, 2015.

[110] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha P. Talukdar. Composition-based multi-relational graph convolutional networks. In 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net, 2020.

[111] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. Advances in neural information processing systems, 30, 2017.

[112] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. stat, 1050:20, 2017.

[113] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. OpenReview.net, 2018.

[114] Haorui Wang, Haoteng Yin, Muhan Zhang, and Pan Li. Equivariant and stable positional encoding for more powerful graph neural networks. arXiv preprint arXiv:2203.00199, 2022.

[115] Weiyao Wang, Du Tran, and Matt Feiszli. What makes training multi-modal classification networks hard? In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 12695–12705, 2020.

[116] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. Kgat: Knowledge graph attention network for recommendation. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 950–958, 2019.

[117] Xiyuan Wang, Haotong Yang, and Muhan Zhang. Neural common neighbor with completion for link prediction. arXiv preprint arXiv:2302.00890, 2023.

[118] Zhitao Wang, Yong Zhou, Litao Hong, Yuanhang Zou, Hanjing Su, and Shouzhi Chen. Pairwise learning for neural link prediction. arXiv preprint arXiv:2112.02936, 2021.

[119] Wei Wei, Xubin Ren, Jiabin Tang, Qinyong Wang, Lixin Su, Suqi Cheng, Junfeng Wang, Dawei Yin, and Chao Huang. Llmrec: Large language models with graph augmentation for recommendation. arXiv preprint arXiv:2311.00423, 2023.

[120] Hongzhi Wen, Jiayuan Ding, Wei Jin, Yiqi Wang, Yuying Xie, and Jiliang Tang. Graph neural networks for multimodal single-cell data integration. In Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining, pages 4153–4163, 2022.

[121] Robert West, Evgeniy Gabrilovich, Kevin Murphy, Shaohua Sun, Rahul Gupta, and Dekang Lin. Knowledge base completion via search-based question answering. In Proceedings of the 23rd international conference on World wide web, pages 515–526, 2014.

[122] Bin Wu, Chenyan Xiong, Maosong Sun, and Zhiyuan Liu. Query suggestion with feedback memory network. In Proceedings of the 2018 World Wide Web Conference, pages 1563–1571, 2018.

[123] Chuhan Wu, Fangzhao Wu, Suyu Ge, Tao Qi, Yongfeng Huang, and Xing Xie. Neural news recommendation with multi-head self-attention. In 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 6389–6394, 2019.

[124] Nan Wu, Stanislaw Jastrzebski, Kyunghyun Cho, and Krzysztof J Geras. Characterizing and overcoming the greedy nature of learning in multi-modal deep neural networks. In International Conference on Machine Learning, pages 24043–24055. PMLR, 2022.

[125] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. Graph neural networks in recommender systems: a survey. ACM Computing Surveys, 55(5):1–37, 2022.

[126] Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. Session-based recommendation with graph neural networks. In Proceedings of the AAAI conference on artificial intelligence, volume 33, pages 346–353, 2019.

[127] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv preprint arXiv:1609.08144, 2016.

[128] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. IEEE transactions on neural networks and learning systems, 32(1):4–24, 2020.

[129] Chenyan Xiong, Russell Power, and Jamie Callan. Explicit semantic ranking for academic search via knowledge graph embedding. In Proceedings of the 26th international conference on world wide web, pages 1271–1279, 2017.

[130] Wenhan Xiong, Thien Hoang, and William Yang Wang. Deeppath: A reinforcement learning method for knowledge graph reasoning. In Martha Palmer, Rebecca Hwa, and Sebastian Riedel, editors, Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017, pages 564–573. Association for Computational Linguistics, 2017.

[131] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015.

[132] Hebin Yang, Qinmin Hu, and Liang He. Learning topic-oriented word embedding for query classification. In Pacific-Asia Conference on Knowledge Discovery and Data Mining, pages 188–198. Springer, 2015.

[133] Yang Yang, Ryan N Lichtenwalter, and Nitesh V Chawla. Evaluating link prediction methods. Knowledge and Information Systems, 45(3):751–782, 2015.

[134] Zhen Yang, Ming Ding, Chang Zhou, Hongxia Yang, Jingren Zhou, and Jie Tang. Understanding negative sampling in graph representation learning. In Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining, pages 1666–1676, 2020.

[135] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In International conference on machine learning, pages 40–48. PMLR, 2016.

[136] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. arXiv preprint arXiv:1906.08237, 2019.

[137] Liang Yao, Chengsheng Mao, and Yuan Luo. Graph convolutional networks for text classification. In Proceedings of the AAAI conference on artificial intelligence, volume 33, pages 7370–7377, 2019.

[138] Rui Ye, Xin Li, Yujie Fang, Hongyu Zang, and Mingzhong Wang. A vectorized relational graph convolutional network for multi-relational network alignment. In IJCAI, pages 4135–4141, 2019.

[139] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 974–983, 2018.

[140] Donghan Yu, Yiming Yang, Ruohong Zhang, and Yuexin Wu. Knowledge embedding based graph convolutional network. In Proceedings of the Web Conference 2021, pages 1619–1628, 2021.

[141] Wenhao Yu, Lingfei Wu, Yu Deng, Ruchi Mahindru, Qingkai Zeng, Sinem Guven, and Meng Jiang. A technical question answering system with transfer learning. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 92–99, 2020.

[142] Zheng Yuan, Fajie Yuan, Yu Song, Youhua Li, Junchen Fu, Fei Yang, Yunzhu Pan, and Yongxin Ni. Where to go next for recommender systems? id-vs. modality-based recommender models revisited. arXiv preprint arXiv:2303.13835, 2023.

[143] Seongjun Yun, Seoyoon Kim, Junhyun Lee, Jaewoo Kang, and Hyunwoo J Kim. Neo-gnns: Neighborhood overlap-aware graph neural networks for link prediction. Advances in Neural Information Processing Systems, 34:13683–13694, 2021.

[144] Haolan Zhan, Hainan Zhang, Hongshen Chen, Lei Shen, Yanyan Lan, Zhuoye Ding, and Dawei Yin. User-inspired posterior network for recommendation reason generation. In

Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 1937–1940, 2020.

[145] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. Advances in neural information processing systems, 31, 2018.

[146] Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. Labeling trick: A theory of using graph neural networks for multi-node representation learning. Advances in Neural Information Processing Systems, 34:9061–9073, 2021.

[147] Shichang Zhang, Yozen Liu, Yizhou Sun, and Neil Shah. Graph-less neural networks: Teaching old mlps new tricks via distillation. ICLR, 2022.

[148] Tingting Zhang, Pengpeng Zhao, Yanchi Liu, Victor S Sheng, Jiajie Xu, Deqing Wang, Guanfeng Liu, Xiaofang Zhou, et al. Feature-level deeper self-attention network for sequential recommendation. In IJCAI, pages 4320–4326, 2019.

[149] Yongqi Zhang and Quanming Yao. Knowledge graph reasoning with relational digraph. In Proceedings of the ACM Web Conference 2022, pages 912–924, 2022.

[150] Zhanqiu Zhang, Jie Wang, Jieping Ye, and Feng Wu. Rethinking graph convolutional networks in knowledge graph completion. In Proceedings of the ACM Web Conference 2022, pages 798–807, 2022.

[151] Lingxiao Zhao, Wei Jin, Leman Akoglu, and Neil Shah. From stars to subgraphs: Uplifting any gnn with local structure awareness. ICLR, 2022.

[152] Zhi Zheng, Kai Hui, Ben He, Xianpei Han, Le Sun, and Andrew Yates. Bert-qe: contextualized query expansion for document re-ranking. arXiv preprint arXiv:2009.07258, 2020.

[153] Kun Zhou, Hui Wang, Wayne Xin Zhao, Yutao Zhu, Sirui Wang, Fuzheng Zhang, Zhongyuan Wang, and Ji-Rong Wen. S3-rec: Self-supervised learning for sequential recommendation with mutual information maximization. In Proceedings of the 29th ACM international conference on information & knowledge management, pages 1893–1902, 2020.

[154] Sanduo Zhou, Kefei Cheng, and Lijun Men. The survey of large-scale query classification. In AIP Conference Proceedings, volume 1834, page 040045. AIP Publishing LLC, 2017.

[155] Tao Zhou, Linyuan Lü, and Yi-Cheng Zhang. Predicting missing links via local information. The European Physical Journal B, 71:623–630, 2009.

[156] Zhaocheng Zhu, Xinyu Yuan, Louis-Pascal Xhonneux, Ming Zhang, Maxime Gazeau, and Jian Tang. Learning to efficiently propagate for reasoning on knowledge graphs. arXiv preprint arXiv:2206.04798, 2022.

[157] Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. Neural bellman-ford networks: A general graph neural network framework for link prediction. Advances in Neural Information Processing Systems, 34, 2021.

[158] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578, 2016.

[159] Lixin Zou, Shengqiang Zhang, Hengyi Cai, Dehong Ma, Suqi Cheng, Shuaiqiang Wang, Daiting Shi, Zhicong Cheng, and Dawei Yin. Pre-trained language model based ranking in baidu search. In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, pages 4014–4022, 2021.

APPENDIX A

EVALUATION PITFALLS IN UNI-RELATIONAL GRAPH

A.1 Common Neighbor Distribution

In Figure 2.1 we demonstrate the common neighbor (CN) distribution among positive and negative test samples for ogbl-collab, ogbl-ppa, and ogbl-citation2. These results demonstrate that a vast majority of negative samples have no CNs. Since CNs is a typically good heuristic, this makes it easy to identify most negative samples.

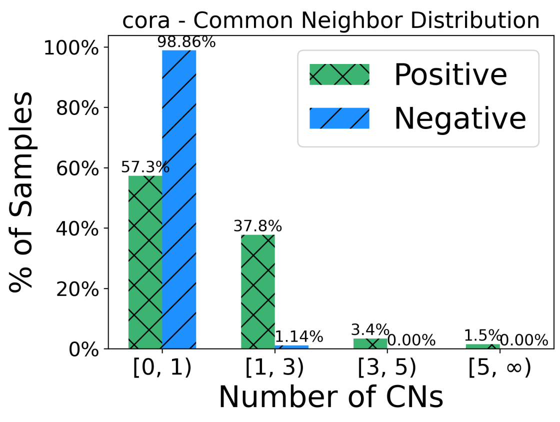
We further present the CN distribution of Cora, Citeseer, Pubmed, and ogbl-ddi in Figure A.1. The CN distribution of Cora, Citeseer, and Pubmed are consistent with our previous observations on the OGB datasets in Figure 2.1. We note that ogbl-ddi exhibits a different distribution with other datasets. As compared to the other datasets, most of the negative samples in ogbl-ddi have common neighbors. This is likely because ogbl-ddi is considerably denser than the other graphs. As shown in Table A.1, the average node degree in ogbl-ddi is 625.68, significantly larger than the second largest dataset ogbl-ppa with 105.25. Thus, despite the random sampling of negative samples, the high degree of node connectivity within the ogbl-ddi graph predisposes a significant likelihood for the occurrence of common neighbors.

We also present the CN distributions under the HeaRT setting. The plots for Cora, Citeseer, Pubmed are shown in Figure A.2. The plots for the OGB datasets are shown in Figure A.3. We observe that the CN distribution of HeaRT is more aligned with the positive samples. This allows for a fairer evaluation setting by not favoring models that use CN information.

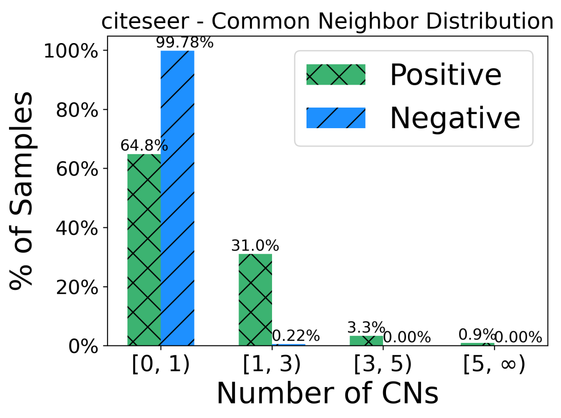
A.2 Additional Definitions

A.2.1 Evaluation Metrics

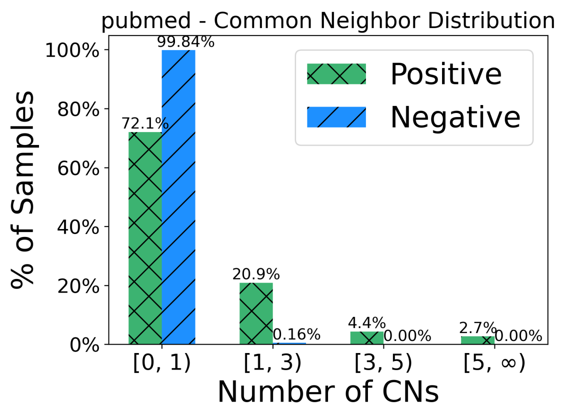
In this section we define the various evaluation metrics used. Given a single positive sample and M negative samples, we first score each sample and then rank the positive sample among the negatives. The rank is then given by rank_i. I.e., a rank of 1 indicates that the positive sample has a higher score than all negatives. The hope is that the positive sample ranks above most or all negative samples. Various metrics make use of this rank. We use N to denote the number of positive



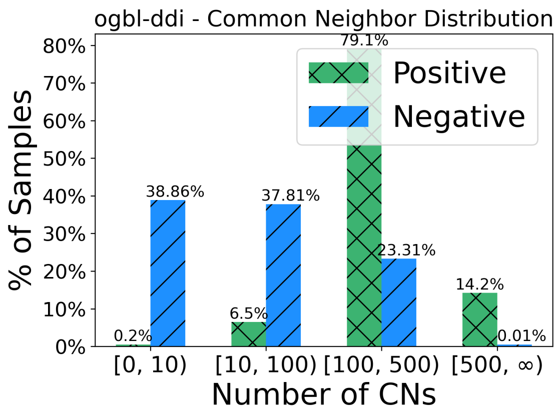
(a) Cora



(b) Citeseer



(c) Pubmed



(d) ogbl-ddi

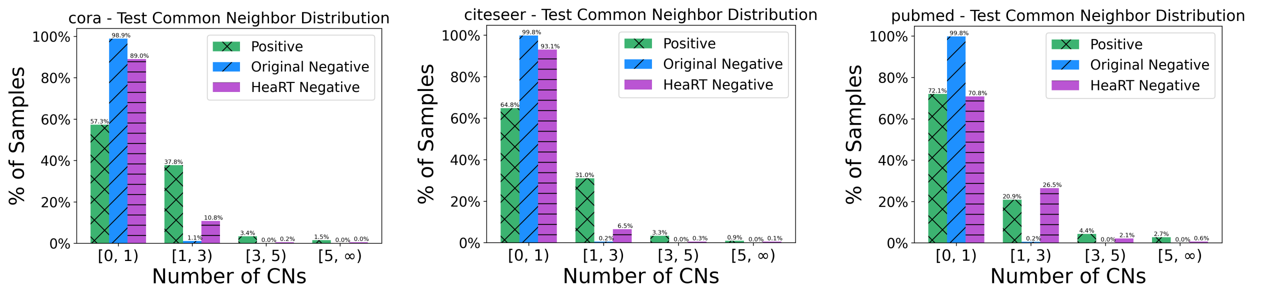
Figure A.1 Common neighbor distribution for the positive and negative test samples for the Cora, Citeseer, Pubmed, and ogbl-ddi under the existing evaluation setting.

samples.

Hits@K. It measures whether the true positive is within the top K predictions or not: Hits@K = $\frac{1}{N}\sum_{i=1}^{N}\mathbf{1}(\text{rank}_i \leq \text{K})$. rank_i is the rank of the i-th sample. The indicator function $\mathbf{1}$ is 1 if $\text{rank}_i \leq \text{K}$, and 0 otherwise.

Mean Reciprocal Rank (MRR). It is the mean of the reciprocal rank over all positive samples: $\text{MRR} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{\text{rank}_i}$, where rank_i is the rank of the i-th sample.

AUC. It measures the likelihood that a positive sample is ranked higher than a random negative sample: $\text{AUC} = \frac{\sum_{i\in\mathcal{D}^0}\sum_{j\in\mathcal{D}^1}\mathbf{1}(\text{rank}_i<\text{rank}_j)}{|\mathcal{D}^0|\cdot|\mathcal{D}^1|}$, where $\mathcal{D}^0$ is the set of positive samples, $\mathcal{D}^1$ is the set of negative samples, and rank_i is the rank of the i-th sample. The indicator function $\mathbf{1}$ is 1 if $\text{rank}_i < \text{rank}_j$, and 0 otherwise.



(a) Cora (b) Citeseer (c) Pubmed

Figure A.2 Common neighbor distribution for the positive negative samples under both evaluation settings for Cora, Citeseer, Pubmed.

A.2.2 Negative Sampling

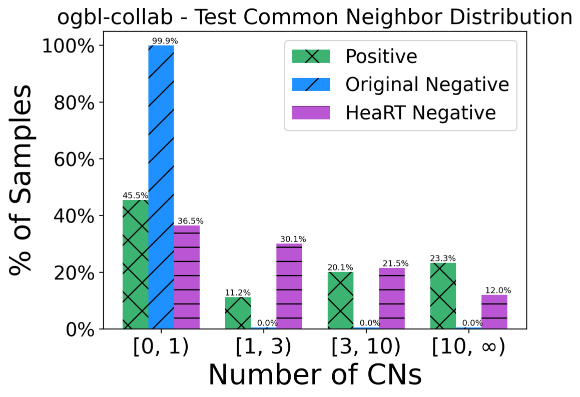
Since only positive links are observed, there is a need to generate negative links (i.e., edges that don't exist in $\mathcal{G}$) to both train and evaluate different models. We detail how these samples are generated in both training and evaluation.

Training Negative Samples. During training, the negative samples are randomly selected, with all nodes being equally likely to be selected. Let $\mathcal{V}$ and $\mathcal{E}$ be the set of nodes and edges in $\mathcal{G}$. Furthermore, we define $v \in \text{Rand}(\mathcal{V})$ as returning a random node in $\mathcal{V}$. A single negative sample (a^-, b^-) is given by:

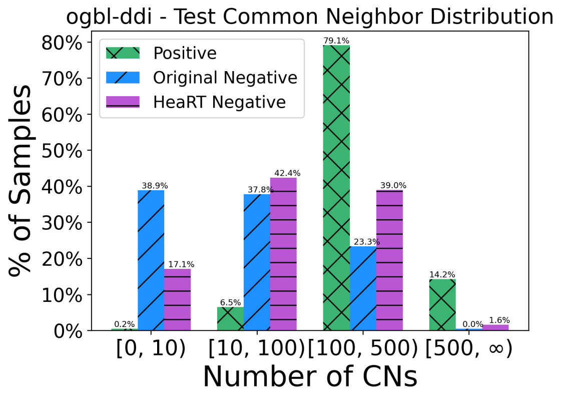
$$(a^-, b^-) = \left(\text{Rand}(\mathcal{V}), \text{Rand}(\mathcal{V})\right). \tag{A.1}$$

Typically one negative sample is generated per positive sample.

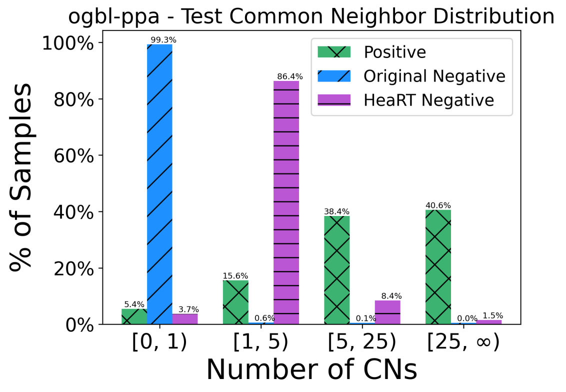
Evaluation Negative Samples. For the existing setting, a fixed set of randomly selected samples are used as negatives during evaluation. Furthermore, the same set of negative samples are used for each positive sample. This is equivalent to Eq. (A.1). The only exception is the ogbl-citation2 [39] dataset. For ogbl-citation2, each positive sample is only evaluated against its own set 1000 negative samples. For a positive sample, its negative samples are restricted to contain one of its two nodes (i.e., a corruption). The other node is randomly selected from $\mathcal{V}$. This is equivalent to selecting a set of random samples from the set $S(a, b)$ as defined in Eq. (2.3).



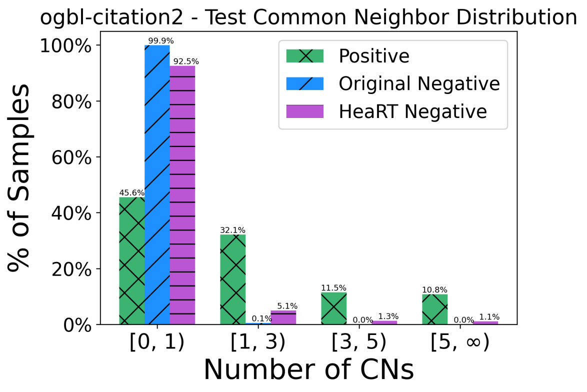
(a) ogbl-collab



(b) ogbl-ddi



(c) ogbl-ppa



(d) ogbl-citation

Figure A.3 Common neighbor distribution for the positive negative samples under both evaluation settings for the OGB datasets.

Table A.1 Statistics of datasets. The split ratio is for train/validation/test. #Nodes: number of nodes, #Edges: number of edges.

	Cora	Citeseer	Pubmed	ogbl-collab	ogbl-ddi	ogbl-ppa	ogbl-citation2
#Nodes	2,708	3,327	18,717	235,868	4,267	576,289	2,927,963
#Edges	5,278	4,676	44,327	1,285,465	1,334,889	30,326,273	30,561,187
Mean Degree	3.9	2.81	4.74	10.90	625.68	105.25	20.88
Split Ratio	85/5/10	85/5/10	85/5/10	92/4/4	80/10/10	70/20/10	98/1/1

A.3 Datasets and Experimental Settings

A.3.1 Datasets

The statistics of datasets are shown in Table A.1. Generally, Cora, Citeseer, and Pubmed are smaller graphs, with the OGB datasets having more nodes and edges. We adopt the single fixed train/validation/test split with percentages 85/5/10% for Cora, Citeseer, and Pubmed. For OGB datasets, we use the fixed splits provided by the OGB benchmark [39].

A.3.2 Experimental Settings

Training Settings. We use the binary cross entropy loss to train each model. The loss is optimized using the Adam optimizer [51]. During training we randomly sample one negative sample per positive sample. Each model is trained for a maximum of 9999 epochs, with the process set to terminate when there are no improvements observed in the validation performance over n checkpoints. The choice of n is influenced by both the specific dataset and the complexity of the model. For smaller datasets, such as Cora, Citeseer, and Pubmed, we set $n = 50$ uniformly across models (except for NBFNet and SEAL where $n = 20$ due to their computational inefficiency). When training on larger OGB datasets, we use a stratified approach: $n = 100$ for the simpler methods (i.e., the embedding and GNN-based models) and $n = 20$ for the more advanced methods. This is due to the increased complexity and runtime of more advanced methods. An exception is for ogbl-citation2, the largest dataset. To accommodate for its size, we limit the maximum number of epochs to the recommended value from each model's source code. Furthermore, we set $n = 20$ for all models.

In order to accommodate the computational requirements for our extensive experiments, we harness a variety of high-capacity GPU resources. This includes: Tesla V100 32Gb, NVIDIA RTX A6000 48Gb, NVIDIA RTX A5000 24Gb, and Quadro RTX 8000 48Gb.

Table A.2 Hyperparameter search ranges on all datasets we used. #Model Layers: number of model layers, #Prediction Layers: number of prediction layers.

Dataset	Learning Rate	Dropout	Weight Decay	# Model Layers	# Prediction Layers	Embedding Dim
Cora	(0.01, 0.001)	(0.1, 0.3, 0.5)	(1e-4, 1e-7, 0)	(1, 2, 3)	(1, 2, 3)	(128, 256)
Citeseer	(0.01, 0.001)	(0.1, 0.3, 0.5)	(1e-4, 1e-7, 0)	(1, 2, 3)	(1, 2, 3)	(128, 256)
Pubmed	(0.01, 0.001)	(0.1, 0.3, 0.5)	(1e-4, 1e-7, 0)	(1, 2, 3)	(1, 2, 3)	(128, 256)
ogbl-collab	(0.01, 0.001)	(0, 0.3, 0.5)	0	3	3	256
ogbl-ddi	(0.01, 0.001)	(0, 0.3, 0.5)	0	3	3	256
ogbl-ppa	(0.01, 0.001)	(0, 0.3, 0.5)	0	3	3	256
ogbl-citation2	(0.01, 0.001)	(0, 0.3, 0.5)	0	3	3	128

Hyperparameter Settings. We present the hyparameter searching range in Table A.2. For the smaller graphs, Cora, Citeseer, and Pubmed, we have a larger search space. However, it's not feasible to tune over such large space for OGB datasets. By following the most commonly used settings among published hyperparameters, we fix the weight decay, number of model and prediction layers, and the embedding dimension. Furthermore, due to GPU memory constraints, the embedding size is reduced to be 128 for the largest dataset ogbl-citation2.

We note that several exceptions exist to these ranges when they result in significant performance degradations. In such instances, adjustments are guided by the optimal hyperparameters published in the respective source codes. This includes:

- PEG [114]: Adhering to the optimal hyperparameters presented in the source code,[1] when training on ogbl-ddi we set the number of model layers to 2 and the maximum number of epochs to 400.
- NCN/NCNC [117]: When training on ogbl-ddi, we adhere to the suggested optimal hyperparameters used in the source code.[2] Specifically, we set the number of model layers to be 1, and we don't apply the pretraining for NCNC to facilitate a fair comparison.
- NBFNet [157]: Due to the expensive nature of NBFNet, we further fix the weight decay to 0 when training on Cora, Citeseer, and Pubmed. Furthermore, we follow the suggested hyperparameters [3] and set the embedding dimension to be 32 and the number of model layers to be 6.
- SEAL [145]: Due to the computational inefficiency of SEAL, when training on Cora, Citeseer and Pubmed we further fix the weight decay to 0. Furthermore, we adhere to the published hyperparameters [4] and fix the number of model layers to be 3 and the embedding dimension to be 256.
- BUDDY [15]: When training on ogbl-ppa, we incorporate the RA and normalized degree as input features while excluding the raw node features. This is based on the optimal hyperparameters published by the authors.[5]

A.4 Reported vs. Our Results on ogbl-ddi

In Section 2.3 (see observation 2), we remarked that there is divergence between the reported results and our results on ogbl-ddi for some methods. A comprehensive comparison of this discrepancy is shown in Table A.3. The reported results for Node2Vec, MF, GCN, and SAGE are

[1]https://github.com/Graph-COM/PEG/
[2]https://github.com/GraphPKU/NeuralCommonNeighbor/
[3]https://github.com/DeepGraphLearning/NBFNet/
[4]https://github.com/facebookresearch/SEAL_OGB/
[5]https://github.com/melifluos/subgraph-sketching/

taken from [39]. The results for the other methods are from their original paper: SEAL [146], BUDDY [15], Neo-GNN [143], NCN [117], NCNC [117], and PEG [114].

Table A.3 Comparison results between ours and reported results on ogbl-ddi (Hits@20). Most of baseline results are under-reported.

	Node2Vec	MF	GCN	SAGE	SEAL	BUDDY	Neo-GNN	NCN	NCNC	PEG
Reported	23.26 ± 2.09	13.68 ± 4.75	37.07 ± 5.07	**53.90 ± 4.74**	**30.56 ± 3.86**	**78.51 ± 1.36**	**63.57 ± 3.52**	**82.32 ± 6.10**	**84.11 ± 3.67**	**43.80 ± 0.32**
Ours	**34.69 ± 2.90**	**23.50 ± 5.35**	**49.90 ± 7.23**	49.84 ± 15.56	25.25 ± 3.90	29.60 ± 4.75	20.95 ± 6.03	76.52 ± 10.47	70.23 ± 12.11	30.28 ± 4.92

A.5 Additional Investigation on ogbl-ddi

In this section, we present the additional investigation on the ogbl-ddi dataset. In Section A.5.1 we examine under the existing evaluation setting, there exists a poor relationship between the validation and test performance on ogbl-ddi for many methods. We then demonstrate in Section A.5.2 that under HeaRT this problem is lessened.

A.5.1 Existing Evaluation Setting

Upon inspection, we found that there is a poor relationship between the validation and test performance on ogbl-ddi. Since we choose the best hyperparameters based on the validation set, this makes it difficult to properly tune any model on ogbl-ddi. To demonstrate this point, we record the validation and test performance at multiple checkpoints during the training process. The experiments are conducted over 10 seeds. To ensure that our results are not caused by our hyperparameter settings, we use the reported hyperparameters for each model. Lastly, we plot the results for GCN, BUDDY, NCN, and Neo-GNN in Figure A.4. It is clear from the results that there exists a poor relationship between the validation and test performance. For example, for NCN, a validation performance of 70 can imply a test performance of 3 to 80. Further investigation is needed to uncover the cause of this misalignment.

A.5.2 New Evaluation Setting

Under our new setting, we find that the validation and test performance have a much better relationship. In Section A.5.1 we observed that there exists a poor relationship between the validation and test performance on ogbl-ddi under the existing evaluation setting. This meddles with our ability to choose the best hyperparameters for each model, as good validation performance is not indicative of good test performance. However, this does not seem to be the case under the new evaluation

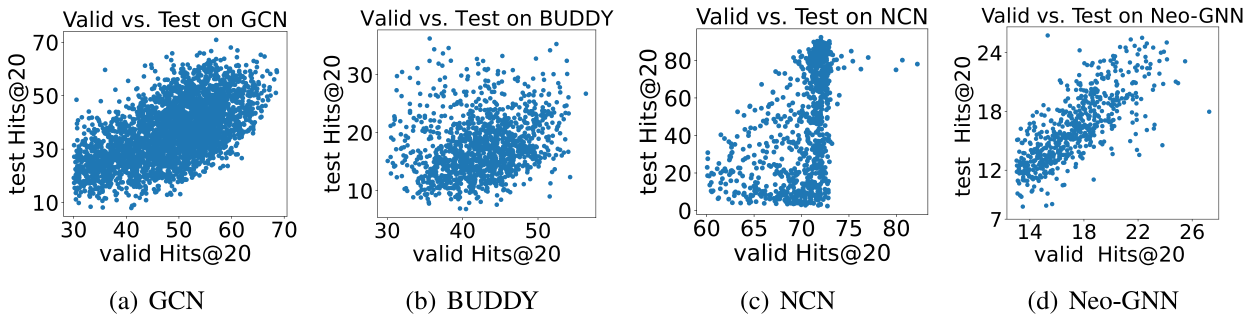


Figure A.4 Validation vs. test performance for GCN, BUDDY, NCN and Neo-GNN on ogbl-ddi under the existing evaluation setting.

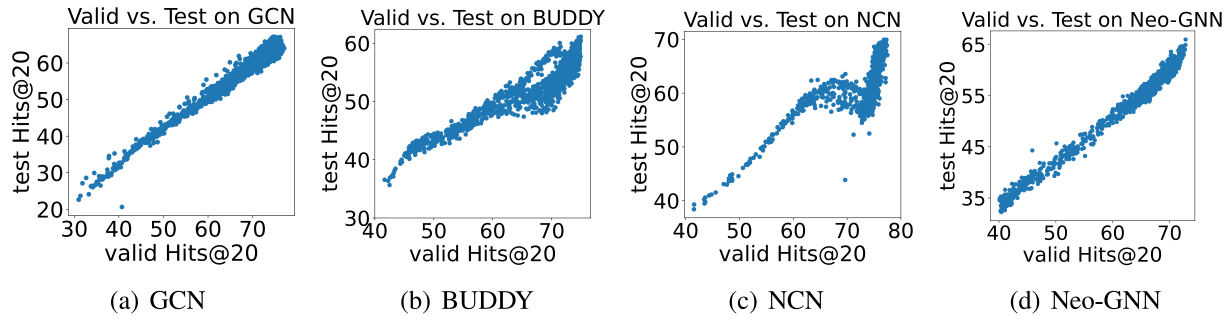


Figure A.5 Validation vs. test performance when utilizing HeaRT for GCN, BUDDY, NCN and Neo-GNN on ogbl-ddi. We find they have a much stronger relationship than under the existing setting (see Figure A.4).

setting. In Figure A.5 we plot the relationship between the validation and test performance by checkpoint for various models. Compared to the same plots under the existing setting (Figure A.4), the new results display a much better relationship.

While it's unclear what is the cause of the poor relationship between the test and validation performance under the existing setting, we conjecture that tailoring the negatives to each positive sample allows for a more natural comparison between a positive sample and its negatives. This may help produce more stable evaluation metrics, thereby strengthening the alignment between the validation and test performance.

A.6 Additional Results Under the Existing Setting

We present additional results of Cora, Citeseer, Pubmed and OGB datasets in Tables A.4-A.7 under the existing setting. We also omit the MRR for ogbl-collab, ogbl-ddi, and ogbl-ppa. This is because the large number of negative samples make it very inefficient to calculate.

Table A.4 Additional results on Cora(%) under the existing evaluation setting. Highlighted are the results ranked first (green), second (blue), and third (orange).

	Models	Hits@1	Hits@3	Hits@10	Hits@100
Heuristic	CN	13.47	13.47	42.69	42.69
	AA	22.2	39.47	42.69	42.69
	RA	20.11	39.47	42.69	42.69
	Shortest Path	0	0	42.69	71.35
	Katz	19.17	28.46	51.61	74.57
Embedding	Node2Vec	**22.3 ± 11.76**	**41.63 ± 10.5**	62.34 ± 2.35	84.88 ± 0.96
	MF	7.76 ± 5.61	13.26 ± 4.52	29.16 ± 6.68	66.39 ± 5.03
	MLP	18.79 ± 11.40	35.35 ± 10.71	53.59 ± 3.57	85.52 ± 1.44
GNN	GCN	16.13 ± 11.18	32.54 ± 10.83	66.11 ± 4.03	91.29 ± 1.25
	GAT	18.02 ± 8.96	**42.28 ± 6.37**	63.82 ± 2.72	90.70 ± 1.03
	SAGE	**29.01 ± 6.42**	**44.51 ± 6.57**	63.66 ± 4.98	91.00 ± 1.52
	GAE	17.57 ± 4.37	24.82 ± 4.91	**70.29 ± 2.75**	**92.75 ± 0.95**
GNN+heuristic	SEAL	12.35 ± 8.57	38.63 ± 4.96	55.5 ± 3.28	84.76 ± 1.6
	BUDDY	12.62 ± 6.69	29.64 ± 5.71	59.47 ± 5.49	91.42 ± 1.26
	Neo-GNN	4.53 ± 1.96	33.36 ± 9.9	64.1 ± 4.31	87.76 ± 1.37
	NCN	19.34 ± 9.02	38.39 ± 7.01	**74.38 ± 3.15**	**95.56 ± 0.79**
	NCNC	9.79 ± 4.56	34.31 ± 8.87	**75.07 ± 1.95**	**95.62 ± 0.84**
	NBFNet	**29.94 ± 5.78**	38.29 ± 3.03	62.79 ± 2.53	88.63 ± 0.46
	PEG	5.88 ± 1.65	30.53 ± 6.42	62.49 ± 4.05	91.42 ± 0.8

Table A.5 Additional results on Citeseer(%) under the existing evaluation setting. Highlighted are the results ranked first (green), second (blue), and third (orange).

	Models	Hits@1	Hits@3	Hits@10	Hits@100
Heuristic	CN	13.85	35.16	35.16	35.16
	AA	21.98	35.16	35.16	35.16
	RA	18.46	35.16	35.16	35.16
	Shortest Path	0	53.41	56.92	62.64
	Katz	24.18	54.95	57.36	62.64
Embedding	Node2Vec	30.24 ± 16.37	54.15 ± 6.96	68.79 ± 3.05	89.89 ± 1.48
	MF	19.25 ± 6.71	29.03 ± 4.82	38.99 ± 3.26	59.47 ± 2.69
	MLP	30.22 ± 10.78	56.42 ± 7.90	69.74 ± 2.19	91.25 ± 1.90
GNN	GCN	37.47 ± 11.30	62.77 ± 6.61	74.15 ± 1.70	91.74 ± 1.24
	GAT	34.00 ± 11.14	62.72 ± 4.60	74.99 ± 1.78	91.69 ± 2.11
	SAGE	27.08 ± 10.27	65.52 ± 4.29	78.06 ± 2.26	**96.50 ± 0.53**
	GAE	**54.06 ± 5.8**	65.3 ± 2.54	**81.72 ± 2.62**	95.17 ± 0.5
GNN+heuristic	SEAL	31.25 ± 8.11	46.04 ± 5.69	60.02 ± 2.34	85.6 ± 2.71
	BUDDY	**49.01 ± 15.07**	**67.01 ± 6.22**	**80.04 ± 2.27**	95.4 ± 0.63
	Neo-GNN	41.01 ± 12.47	59.87 ± 6.33	69.25 ± 1.9	89.1 ± 0.97
	NCN	35.52 ± 13.96	66.83 ± 4.06	79.12 ± 1.73	**96.17 ± 1.06**
	NCNC	**53.21 ± 7.79**	**69.65 ± 3.19**	**82.64 ± 1.4**	**97.54 ± 0.59**
	NBFNet	17.25 ± 5.47	51.87 ± 2.09	68.97 ± 0.77	86.68 ± 0.42
	PEG	39.19 ± 8.31	**70.15 ± 4.3**	77.06 ± 3.53	94.82 ± 0.81

Table A.6 Additional results on Pubmed(%) under the existing evaluation setting. Highlighted are the results ranked first (green), second (blue), and third (orange).

	Models	Hits@1	Hits@3	Hits@10	Hits@100
Heuristic	CN	7.06	12.95	27.93	27.93
	AA	12.95	16	27.93	27.93
	RA	11.67	15.21	27.93	27.93
	Shortest Path	0	0	27.93	60.36
	Katz	12.88	25.38	42.17	61.8
Embedding	Node2Vec	**29.76 ± 4.05**	34.08 ± 2.43	44.29 ± 2.62	63.07 ± 0.34
	MF	12.58 ± 6.08	22.51 ± 5.6	32.05 ± 2.44	53.75 ± 2.06
	MLP	7.83 ± 6.40	17.23 ± 2.79	34.01 ± 4.94	84.19 ± 1.33
GNN	GCN	5.72 ± 4.28	19.82 ± 7.59	56.06 ± 4.83	87.41 ± 0.65
	GAT	6.45 ± 10.37	23.02 ± 10.49	46.77 ± 4.03	80.95 ± 0.72
	SAGE	11.26 ± 6.86	27.23 ± 7.48	48.18 ± 4.60	**90.02 ± 0.70**
	GAE	1.99 ± 0.12	31.75 ± 1.13	45.48 ± 1.07	84.3 ± 0.31
GNN+heuristic	SEAL	**30.93 ± 8.35**	**40.58 ± 6.79**	48.45 ± 2.67	76.06 ± 4.12
	BUDDY	15.31 ± 6.13	29.79 ± 6.76	46.62 ± 4.58	83.21 ± 0.59
	Neo-GNN	19.95 ± 5.86	34.85 ± 4.43	**56.25 ± 3.42**	86.12 ± 1.18
	NCN	26.38 ± 6.54	**36.82 ± 6.56**	**62.15 ± 2.69**	**90.43 ± 0.64**
	NCNC	9.14 ± 5.76	33.01 ± 6.28	**61.89 ± 3.54**	**91.93 ± 0.6**
	NBFNet	**40.47 ± 2.91**	**44.7 ± 2.58**	54.51 ± 0.84	79.18 ± 0.71
	PEG	8.52 ± 3.73	24.46 ± 6.94	45.11 ± 4.02	76.45 ± 3.83

Table A.7 Additional results on OGB datasets(%) under the existing evaluation setting. Highlighted are the results ranked first (green), second (blue), and third (orange).

	ogbl-collab		ogbl-ddi		ogbl-ppa		ogbl-citation2		
	Hits@20	Hits@100	Hits@50	Hits@100	Hits@20	Hits@50	Hits@20	Hits@50	Hits@100
CN	49.98	65.6	26.51	34.52	13.26	19.67	77.99	77.99	77.99
AA	55.79	65.6	27.07	36.35	14.96	21.83	77.99	77.99	77.99
RA	55.01	65.6	19.14	31.17	25.64	**38.81**	77.99	77.99	77.99
Shortest Path	46.49	66.82	0	0	0	0	>24h	>24h	>24h
Katz	**58.11**	**71.04**	26.51	34.52	13.26	19.67	78	78	78
Node2Vec	40.68 ± 1.75	55.58 ± 0.77	59.19 ± 3.61	73.49 ± 3.18	11.22 ± 1.91	19.22 ± 1.69	82.8 ± 0.13	92.33 ± 0.1	96.44 ± 0.03
MF	39.99 ± 1.25	43.22 ± 1.94	45.51 ± 11.13	61.72 ± 6.56	9.33 ± 2.83	21.08 ± 3.92	70.8 ± 12.0	74.48 ± 10.42	75.5 ± 10.13
MLP	27.66 ± 1.61	42.13 ± 1.09	N/A	N/A	0.16 ± 0.0	0.26 ± 0.03	74.16 ± 0.1	86.59 ± 0.08	93.14 ± 0.06
GCN	44.92 ± 3.72	62.67 ± 2.14	74.54 ± 4.74	85.03 ± 3.41	11.17 ± 2.93	21.04 ± 3.11	**98.01 ± 0.04**	**99.03 ± 0.02**	**99.48 ± 0.02**
GAT	43.59 ± 4.17	62.24 ± 2.29	55.46 ± 10.16	69.74 ± 10.01	OOM	OOM	OOM	OOM	OOM
SAGE	50.77 ± 2.33	65.36 ± 1.05	**93.48 ± 1.36**	**97.37 ± 0.55**	19.37 ± 2.65	31.3 ± 2.36	97.48 ± 0.03	98.75 ± 0.03	99.3 ± 0.02
GAE	OOM	OOM	12.39 ± 8.74	14.03 ± 9.22	OOM	OOM	OOM	OOM	OOM
SEAL	54.19 ± 1.57	69.94 ± 0.72	43.34 ± 3.23	52.2 ± 1.78	21.81 ± 4.3	36.88 ± 4.06	94.61 ± 0.11	95.0 ± 0.12	95.37 ± 0.14
BUDDY	**57.78 ± 0.59**	67.87 ± 0.87	53.36 ± 2.57	71.04 ± 2.56	**26.33 ± 2.63**	38.18 ± 1.32	**97.79 ± 0.07**	**98.86 ± 0.04**	99.38 ± 0.03
Neo-GNN	**57.05 ± 1.56**	**71.76 ± 0.55**	33.88 ± 10.1	46.55 ± 13.29	26.16 ± 1.24	37.95 ± 1.45	97.05 ± 0.07	98.75 ± 0.03	**99.41 ± 0.02**
NCN	50.27 ± 2.72	67.58 ± 0.09	**95.51 ± 0.87**	**97.54 ± 0.7**	**40.29 ± 2.22**	**53.35 ± 1.77**	**97.97 ± 0.03**	**99.02 ± 0.02**	**99.5 ± 0.01**
NCNC	54.91 ± 2.84	**70.91 ± 0.25**	**92.34 ± 2.42**	**96.35 ± 0.52**	**40.1 ± 1.06**	**52.09 ± 1.99**	97.22 ± 0.78	98.2 ± 0.71	98.77 ± 0.6
NBFNet	OOM	OOM	>24h	>24h	OOM	OOM	OOM	OOM	OOM
PEG	33.57 ± 7.40	55.14 ± 2.10	47.93 ± 3.18	59.95 ± 2.52	OOM	OOM	OOM	OOM	OOM

A.7 Additional Details on HeaRT

As described in Section 2.4.2, given a positive sample (a, b), we seeks to generate K negative samples to evaluate against. The negative samples are drawn from the set of possible corruptions of (a, b), i.e, $S(a, b)$ (see Eq. (2.3)). Multiple heuristics are used to determine which K negative samples to use. Furthermore, the negative samples are split evenly between both nodes. That is, we generate $K/2$ negative samples that contain either node a and b, respectively. This process is illustrated in Figure 2.2. The rest of this section is structured as follows. In Section A.7.1 we describe how we use multiple heuristics for estimating the difficulty of negative samples. Then in Section A.7.2 we describe how we combine the ranks given by different heuristic methods.

A.7.1 Determining Hard Negative Samples

We are first tasked with *how to choose the negative samples*. As discussed and shown in Section 2.4.2, we want to select the negative samples from $S(a, b)$ such that they are non-trivial to classify. Hence, as inspired by the candidate generation process in real-world recommender systems [32, 26], we aim to select a set of 'hard' negative samples that are more relevant to the source node. The candidate generation process is typically based on some primitive and simple link prediction heuristics. These heuristics can be also treated as link prediction methods (see Tables 2.1 and 2.2).

We use multiple heuristics that capture a variety of different information. Most link prediction heuristics can be categorized into two main categories: local heuristics and global heuristics [71]. Local heuristics attempt to capture the local neighborhood information that exists near the node pair while global heuristics attempt to use the whole graph structure. To capture the *local* information we use resource allocation (RA) [155], a CN-based approach. Existing results show that RA can achieve strong performance on most datasets (see Tables 2.1 and 2.2). To measure the *global* information we use the personalized pagerank score (PPR) [9]. Random walk based methods are commonly used for candidate generation [32, 26]. Lastly, we further include the cosine feature similarity for the Cora, Citeseer, and Pubmed datasets. This is due to the strong performance of a MLP on those datasets. By combining these heuristics, we are able to generate a diverse set of negative samples for each positive sample.

For each heuristic we then rank all the possible negative samples. We first denote the score of a heuristic i for a pair of nodes a and b as $h_i(a, b)$. Let's say we want to rank all negative samples that contain a node a, i.e., $(a, *)$. The rank across all nodes is given by:

$$R_i = \underset{v \in \bar{V}}{\text{ArgSort}}\ h_i(a, v), \tag{A.2}$$

where R_i denotes the ranking for heuristic i and and $\bar{V}$ is a subset of the set of nodes in the graph V. We note that all training samples, self-loops, and the sample itself are not able to be chosen as negative samples. Furthermore, when choosing negative samples for the test samples, we disallow validation samples to be chosen as well. As such, we only consider a subset of nodes $\bar{V} \in V$. This is analogous to the filtered setting used in KGC [7].

We now have all possible negative samples ranked according to multiple heuristics. However, it is unclear how to choose the negative samples from multiple ranked lists. In the next subsection we detail how we combine the ranks according to each heuristic. This will give us a final ranking, of which we can choose the top $K/2$ as the negative samples for that node.

A.7.2 Combining Heuristic Ranks

Algorithm A.1 Generating Negative Samples of Form $(a, *)$

```
Require:
    $a$ = Node to generate samples for
    $\bar{V}$ = Possible nodes to use for negative samples
    $\mathcal{H} = \{h_1, h_2, \cdots, h_m\}$                    ▷ Set of $m$ heuristics

1: for $i \in |\mathcal{H}|$ do
2:     $R_i = \underset{v \in \bar{V}}{\text{ArgSort}}\ h_i(a, v)$          ▷ Sort by each heuristic individually
3: end for
4: for $v \in \bar{V}$ do
5:     $R_{\text{total}}(a, v) = \min\left(R_1(a, v), R_2(a, v), \cdots, R_m(a, v)\right)$     ▷ Combine the rankings
6: end for

7: $R_f = \underset{v \in \bar{V}}{\text{ArgSort}}\ R_{\text{total}}(a, v)$          ▷ Sort by combined ranking
8: return $R_f[: K/2]$                    ▷ Return the top K/2 ranked nodes
```

In this subsection we tackle the problem of combing the negative sample ranks given by multiple heuristics. More concretely, say we use m heuristics and rank all the samples according to each. We want to arrive at a combined ranking R_{total} that is composed of each rank,

$$R_{\text{total}} = \phi(R_1, R_2, \cdots, R_m). \tag{A.3}$$

We model ϕ via Borda's method [6]. Let $R_i(a, v)$ be the rank of the node pair (a, v) for heuristic i. The combined rank $R_{\text{total}}(a, v)$ across m ranked lists is given by:

$$R_{\text{total}}(a, v) = g\left(R_1(a, v), R_2(a, v), \cdots, R_m(a, v)\right), \tag{A.4}$$

where g is an aggregation function. We set $g = \min(\cdot)$. This is done as it allows us to capture a more distinct set of samples by selecting the "best" for each heuristic. This is especially true when there is strong disagreement between the different heuristics. A final ranking is then done on R_{total} to select the top nodes,

$$R_f = \underset{v \in \bar{V}}{\text{ArgSort}}\, R_{\text{total}}, \tag{A.5}$$

where R_f is the final ranking. The highest $K/2$ nodes are then selected from R_f. Lastly, we note that for some nodes there doesn't exist sufficient scores to rank $K/2$ total nodes. In this case the remaining nodes are chosen randomly. The full generation process for a node a is detailed in Algorithm A.1.

A.8 Additional Investigation on ogbl-collab

In Section 2.4.3 we observed that under HeaRT, both the Shortest Path and Katz perform considerably well on ogbl-collab under the HeaRT setting. Specifically, the MRR gap between the second-ranked method (Shortest Path) and the third (BUDDY) is 14.29. Since we do not observe this under the existing setting, we further investigate the reason.

We analyzed the performance of both methods on the ogbl-collab dataset. Interestingly, we find that it is due to the dataset being a dynamic graph. In this dataset, nodes represent authors and edges represent a collaboration between two authors. Each edge further includes an attribute that specifies the year of collaboration. Specifically, each edge takes the form of (Author 1, Year, Author

Table A.8 Results on ogbl-collab under HeaRT with new hard negative samples. Highlighted are the results ranked first (green), second (blue), and third (orange).

	MRR	**Hits20**	**Hits50**	**Hits100**
CN	4.20	16.46	30.52	42.80
AA	5.07	19.59	33.74	45.20
RA	**6.29**	**24.29**	**36.68**	46.42
Shortest Path	2.66	15.98	33.77	45.85
Katz	**6.31**	**24.34**	**39.18**	48.80
Node2Vec	4.68 ± 0.08	16.84 ± 0.17	28.56 ± 0.17	41.84 ± 0.25
MF	4.89 ± 0.25	18.86 ± 0.40	30.83 ± 0.22	43.23 ± 0.34
MLP	5.37 ± 0.14	16.15 ± 0.27	28.88 ± 0.32	46.83 ± 0.33
GCN	6.09 ± 0.38	22.48 ± 0.81	35.29 ± 0.49	**50.83 ± 0.21**
GAT	4.18 ± 0.33	18.30 ± 1.42	32.92 ± 1.41	46.71 ± 0.84
SAGE	5.53 ± 0.5	21.26 ± 1.32	33.48 ± 1.40	48.33 ± 0.49
GAE	OOM	OOM	OOM	OOM
SEAL	**6.43 ± 0.32**	21.57 ± 0.38	33.57 ± 0.84	43.06 ± 1.09
Neo-GNN	5.23 ± 0.9	21.03 ± 3.39	36.11 ± 2.36	**49.25 ± 0.81**
NBFNet	OOM	OOM	OOM	OOM
BUDDY	5.67 ± 0.36	**23.35 ± 0.73**	**39.04 ± 0.11**	**50.49 ± 0.09**
PEG	4.83 ± 0.21	18.29 ± 1.06	30.12 ± 0.63	45.40 ± 0.66
NCN	5.09 ± 0.38	20.84 ± 1.31	34.53 ± 0.98	45.69 ± 0.42
NCNC	4.73 ± 0.86	20.49 ± 3.97	34.96 ± 3.80	46.93 ± 2.04

2). The task is to predict collaborations in 2019 (test) based on those until 2017 (training) and 2018 (validation).

We found that of the positive samples in the test set, around 46% also appear as positive samples in the training set. In particular, an edge (Author 1, 2017, Author 2) in the training data may also "appear" in the test data in the form of (Author 1, 2019, Author 2). This is because two authors who collaborated in the past often tend to collaborate again in the future. As such, when evaluating the test sample (Author 1, 2019, Author 2), there is a path of length 1 between the two authors. Furthermore, this co-occurrence phenomenon is common among positive samples but not observed among negatives. This is because we exclude the positive training samples when generating the negative samples for evaluation. **As a result of this exclusion, the presence of a direct link (i.e., a shortest path of length 1) between two authors suggest a positive sample, while its absence often corresponds to a negative sample.** This explains why methods like Shortest Path and Katz can achieve good performance on ogbl-collab.

To mitigate this issue, we introduce a **new set of hard negatives**, permitting the negative samples to also exhibit a shortest path of 1. We do so by not excluding the positive training samples when generating the negative samples for validation and test. We note that we also do not exclude positive validation edges when generating the negatives for test. In simpler terms, when creating negative samples for testing, both positive samples from training and validation are considered. This means that negative samples during testing could present in the training and validation positive samples. This approach is reasonable and well-aligned with the real-world scenario in the context of collaboration graphs. Specifically, authors who collaborated in the past might not do so in the future. For instance, just because the positive sample (Author 2, 2017, Author 3) exists, it does not imply that (Author 2, 2019, Author 3) is also true. However, our previous setting assumed so.

Results using the new hard negatives are presented in Table A.8. We observe that the Shortest Path no longer demonstrates the best performance. In contrast, both Katz and RA perform notably well, underscoring the significance of the common neighbors information (i.e., paths of length 2). Since Katz considers all paths between two nodes, it might benefit from the effective performance of methods based on common neighbors. Additionally, the overall results are inferior to those using the old hard negatives, as seen in Table 2.5. For instance, the MRR values in Table 2.5 exceed 10, the highest MRR in Table A.8 is just approximately 6, suggesting the increased difficulty introduced by the new hard negative strategy.

A.9 Additional Results Under HeaRT

We present additional results of Cora, Citeseer, Pubmed, and OGB datasets under HeaRT in Table A.9 and Table A.10. These results include other hit metrics not found in the main tables.

A.10 Limitation

One potential limitation of HeaRT lies in the generation of customized negative samples for each positive sample. This design may result in an increased number of negative samples compared to the existing setting. Although this provides a more realistic evaluation, it could have an impact on the efficiency of the evaluation process, especially in scenarios where a significant number of positive samples exist. Nonetheless, this limitation does not detract from the potential benefits of HeaRT in providing a more realistic and meaningful link prediction evaluation setting. Furthermore, as each evaluation node pair is independent, it offers scope for parallelization, mitigating any potential

Table A.9 Additional results on Cora, Citeseer, and Pubmed(%) under HeaRT. Highlighted are the results ranked first (green), second (blue), and third (orange).

	Cora			Citeseer			Pubmed		
Models	Hits1	Hits3	Hits100	Hits1	Hits3	Hits100	Hits1	Hits3	Hits100
CN	3.98	10.25	38.71	2.2	9.45	33.63	0.47	1.49	19.29
AA	5.31	12.71	38.9	3.96	12.09	33.85	0.74	1.87	20.37
RA	5.31	12.52	38.52	4.18	11.21	34.07	0.72	1.78	20.04
Shortest Path	0.57	2.85	55.6	0.22	3.52	53.85	0	0.02	21.57
Katz	4.64	11.95	59.96	3.74	11.87	55.82	0.74	2.12	32.78
Node2Vec	5.69 ± 0.81	15.1 ± 0.99	77.21 ± 2.34	9.63 ± 0.82	23.5 ± 1.37	84.46 ± 1.86	0.75 ± 0.14	2.4 ± 0.32	52.27 ± 0.65
MF	1.46 ± 0.8	5.46 ± 1.67	59.68 ± 3.41	2.68 ± 0.92	7.25 ± 0.98	53.25 ± 2.91	1.13 ± 0.24	3.25 ± 0.44	50.56 ± 1.11
MLP	5.48 ± 0.99	14.15 ± 1.56	77.0 ± 1.02	10.44 ± 0.82	26.46 ± 1.24	86.83 ± 1.36	1.28 ± 0.22	4.33 ± 0.28	76.34 ± 0.79
GCN	**7.59 ± 0.61**	**17.46 ± 0.82**	**85.47 ± 0.52**	9.27 ± 0.99	23.19 ± 0.98	89.1 ± 2.13	2.09 ± 0.31	5.58 ± 0.27	73.59 ± 0.53
GAT	5.03 ± 0.81	13.66 ± 0.67	80.87 ± 1.32	8.02 ± 1.21	20.09 ± 0.82	86.83 ± 1.09	1.14 ± 0.16	3.06 ± 0.36	67.06 ± 0.69
SAGE	5.48 ± 0.97	**15.43 ± 1.07**	81.61 ± 0.96	8.37 ± 1.62	23.74 ± 1.62	**92.33 ± 0.68**	**3.03 ± 0.46**	**8.19 ± 1.0**	**79.47 ± 0.53**
GAE	**9.72 ± 0.73**	**19.24 ± 0.76**	79.66 ± 0.95	**13.81 ± 0.82**	**27.71 ± 1.34**	85.49 ± 1.37	1.48 ± 0.23	4.05 ± 0.39	59.79 ± 0.67
SEAL	3.89 ± 2.04	10.82 ± 4.04	61.9 ± 13.97	5.08 ± 1.31	13.68 ± 1.32	68.94 ± 2.3	1.47 ± 0.32	4.71 ± 0.68	65.81 ± 2.43
BUDDY	5.88 ± 0.60	13.76 ± 1.03	82.46 ± 1.79	10.09 ± 0.50	26.11 ± 1.26	**92.66 ± 0.92**	**2.24 ± 0.17**	5.93 ± 0.21	72.01 ± 0.46
Neo-GNN	5.71 ± 0.41	13.89 ± 0.82	80.28 ± 1.08	6.81 ± 0.73	17.8 ± 1.19	85.51 ± 1.01	1.90 ± 0.24	**6.07 ± 0.47**	**76.57 ± 0.58**
NCN	4.85 ± 0.81	14.46 ± 0.98	**84.14 ± 1.24**	**16.77 ± 2.05**	**30.51 ± 0.97**	90.42 ± 0.98	1.13 ± 0.18	3.95 ± 0.24	71.46 ± 0.97
NCNC	4.78 ± 0.71	14.72 ± 1.24	**85.62 ± 0.83**	**11.14 ± 0.82**	**27.21 ± 0.96**	**92.73 ± 1.16**	**2.73 ± 0.49**	**7.05 ± 0.72**	**79.22 ± 0.96**
NBFNet	5.31 ± 1.16	14.95 ± 0.72	76.24 ± 0.68	5.95 ± 1.06	14.53 ± 1.19	72.66 ± 0.95	>24h	>24h	>24h
PEG	**6.98 ± 0.57**	14.93 ± 0.61	82.52 ± 1.28	9.93 ± 0.6	21.91 ± 0.59	90.15 ± 1.43	0.88 ± 0.18	2.61 ± 0.39	64.95 ± 1.81

Table A.10 Additional results on OGB datasets(%) under HeaRT. Highlighted are the results ranked first (green), second (blue), and third (orange).

	ogbl-collab		ogbl-ddi		ogbl-ppa		ogbl-citation2	
Models	Hits50	Hits100	Hits50	Hits100	Hits50	Hits100	Hits50	Hits100
CN	38.39	47.4	70.12	86.53	80.53	86.51	57.56	68.04
AA	42.61	50.25	71.08	87.36	81.93	87.55	58.87	69.39
RA	46.9	51.78	76.39	90.96	81.65	86.84	58.88	68.83
Shortest Path	**46.97**	48.11	0	0	1.34	1.4	>24h	>24h
Katz	**51.07**	54.28	70.12	86.53	80.53	86.51	54.97	67.56
Node2Vec	35.49 ± 0.22	46.12 ± 0.34	**98.38 ± 0.7**	99.91 ± 0.01	69.94 ± 0.06	81.88 ± 0.06	61.22 ± 0.16	77.11 ± 0.13
MF	43.62 ± 0.08	51.75 ± 0.14	95.52 ± 0.72	99.54 ± 0.08	83.29 ± 3.35	89.75 ± 1.9	29.64 ± 7.3	65.87 ± 8.37
MLP	35.32 ± 0.74	51.09 ± 0.37	N/A	N/A	5.36 ± 0.0	22.01 ± 0.01	61.29 ± 0.07	76.94 ± 0.1
GCN	43.17 ± 0.36	**54.93 ± 0.14**	97.65 ± 0.68	99.85 ± 0.06	81.48 ± 0.48	89.62 ± 0.23	70.77 ± 0.34	**85.43 ± 0.18**
GAT	42.07 ± 1.51	53.45 ± 0.64	98.15 ± 0.24	99.93 ± 0.02	OOM	OOM	OOM	OOM
SAGE	43.02 ± 0.63	**54.38 ± 0.27**	**99.17 ± 0.11**	**99.98 ± 0.01**	81.84 ± 0.24	89.46 ± 0.13	**71.91 ± 0.1**	**85.86 ± 0.09**
GAE	OOM	OOM	28.29 ± 13.65	48.34 ± 15.0	OOM	OOM	OOM	OOM
SEAL	43.5 ± 1.75	49.25 ± 1.39	82.42 ± 3.37	92.63 ± 2.05	**87.34 ± 0.49**	**92.45 ± 0.26**	65.11 ± 2.33	77.64 ± 2.43
BUDDY	**50.57 ± 0.18**	**55.63 ± 0.68**	97.81 ± 0.31	**99.93 ± 0.01**	82.5 ± 0.51	88.36 ± 0.32	67.47 ± 0.32	81.94 ± 0.26
Neo-GNN	46.93 ± 0.17	53.81 ± 0.19	83.45 ± 11.03	94.7 ± 4.82	81.21 ± 1.39	88.31 ± 0.19	62.14 ± 0.51	79.13 ± 0.42
NCN	45.89 ± 1.11	52.36 ± 0.33	**98.43 ± 0.22**	**99.96 ± 0.01**	**89.37 ± 0.28**	**93.11 ± 0.27**	**71.56 ± 0.03**	84.01 ± 0.05
NCNC	44.76 ± 4.64	52.41 ± 2.09	>24h	>24h	**91.0 ± 0.24**	**94.72 ± 0.18**	**72.85 ± 0.9**	**86.35 ± 0.51**
NBFNet	OOM	OOM	>24h	>24h	OOM	OOM	OOM	OOM
PEG	38.71 ± 0.17	49.34 ± 0.70	84.21 ± 9.2	95.76 ± 3.48	OOM	OOM	OOM	OOM

efficiency concerns to a large extent. Future work can investigate ways to optimize this process.

A.11 Social Impact

Our method HeaRT harbors significant potential for positive societal impact. By aligning the evaluation setting more closely with real-world scenarios, it enhances the applicability of link prediction research. This not only contributes to the refinement of existing prediction methods but also stimulates the development of more effective link prediction methods. As link prediction has far-reaching implications across numerous domains, from social network analysis to recommendation systems and beyond, improving its performance and accuracy is of paramount societal importance. We also carefully consider the broader impact from various perspectives such as fairness, security, and harm to people. No apparent risk is related to our work.

APPENDIX B

EVALUATION PITFALLS IN KNOWLEDGE GRAPH

B.1 Dataset

We use five well-known KG datasets – Table B.1 details their statistics:

- **FB15k** [8] is a subset of the Freebase database [5] containing general facts. It is constructed by selecting a subset of entities that are both in the Wikilinks database[1] and Freebase.
- **FB15k-237** [109, 108] is a subset of the FB15k which removes the inverse relations from `FB15K`to prevent direct inference.
- **WN18** [96] is subset of the WordNet database [29] which contains lexical relations between words.
- **WN18RR** [21] is a subset of the WN18. WN18 contains triplets in the test set that are generated by inverting triplets from the training set. WN18RR is constructed by removing these triplets to avoid inverse relation test leakage.
- **NELL-995** [130] is constructed from the 995-th iteration of the NELL system [14] which constantly extracts facts from the web.

B.2 Evaluation Metrics

We use the rank-based measures to evaluate the quality of the prediction including Mean Reciprocal Rank (**MRR**) and **Hits@N**. Their detailed definitions are introduced below:

- Mean Reciprocal Rank (**MRR**) is the mean of the reciprocal predicted rank for the ground-truth entity over all triplets in the test set. A higher MRR indicates better performance.

Table B.1 Data statistics for four datasets. Train edges, Val. edges, Test edges: number of train edges/validation edges/test edges.

Datasets	Entities	Relations	Train edges	Val. edges	Test edges
FB15k-237	14,541	237	272,115	17,535	20,466
WN18RR	40,943	11	86,835	3,034	3,134
WN18	40,943	18	141,442	5,000	5,000
FB15k	14,951	1,345	483,142	50,000	59, 071
NELL-995	75.492	200	126,176	13,912	14,125

[1]https://code.google.com/archive/p/wiki-links/

- **Hits@N** calculates the proportion of the groundtruth tail entities with a rank smaller or equal to N over all triplets in the test set. Similar to MRR, a higher Hits@N indicates better performance.

These metrics are indicative, but they can be flawed when a tuple (i.e., (h, r) or (r, t)) has multiple ground-truth entities which appear in either the training, validation or test sets. Following the filtered setting in previous works [8, 96, 110], we remove the misleading entities when ranking and report the filtered results.

B.3 Message Passing in MPNN-based KGC

For a general triplet (h, r, t), we use $\mathbf{x}_h^{(k)}$, $\mathbf{x}_r^{(k)}$, and $\mathbf{x}_t^{(k)}$ to denote the head, relation, and tail embeddings obtained after the k-th layer. Specifically, the input embeddings of the first layer $\mathbf{x}_h^{(0)}$, $\mathbf{x}_r^{(0)}$ and $\mathbf{x}_t^{(0)}$ are randomly initialized. Next, we describe the information aggregation process in the $(k+1)$-th layer for the studied three MPNN-based models, i.e., CompGCN, RGCN and KBGAT.

- **RGCN** [96] aggregates neighborhood information with the relation-specific transformation matrices:

$$\mathbf{x}_h^{(k+1)} = g\Big(\sum_{(r,t)\in\mathcal{N}_h} \frac{1}{c_{h,r}} \mathbf{W}_r^{(k)} \mathbf{x}_t^{(k)} + \mathbf{W}_o^{(k)} \mathbf{x}_h^{(k)}\Big) \tag{B.1}$$

  where $\mathbf{W}_o^{(k)} \in \mathbb{R}^{d_{k+1}\times d_k}$ and $\mathbf{W}_r^{(k)} \in \mathbb{R}^{d_{k+1}\times d_k}$ are learnable matrices. $\mathbf{W}_r^{(k)}$ corresponds to the relation r, $\mathcal{N}_h$ is the set of neighboring tuples (r, t) for entity h, g is a non-linear function, and $c_{h,r}$ is a normalization constant that can be either learned or predefined.

- **CompGCN** [110] defines direction-based transformation matrices and introduce relation embeddings to aggregate the neighborhood information:

$$\mathbf{x}_t^{(k+1)} = g\left(\sum_{(h,r)\in\mathcal{N}_t} \mathbf{W}_{\lambda(r)}^{(k)}\, \phi(\mathbf{x}_h^{(k)}, \mathbf{x}_r^{(k)})\right), \tag{B.2}$$

  where $\mathcal{N}_t$ is the set of neighboring entity-relation tuples (h, r) for entity t, $\lambda(r)$ denotes the direction of relations: original relation, inverse relation, and self-loop. $\mathbf{W}_{\lambda(r)}^{(k)} \in \mathbb{R}^{d_{k+1}\times d_k}$ is the direction specific learnable weight matrix in the k-th layer, and $\phi(\cdot)$ is the composition operator to combine the embeddings to leverage the entity-relation information. The composition operator $\phi(\cdot)$ is defined as the subtraction, multiplication, or cross correlation of the two embeddings [110]. CompGCN generally achieves best performance when adopting the cross correlation.

Hence, in this work, we use the cross correlation as its default composition operation for our investigation. CompGCN updates the relation embedding through linear transformation in each layer, i.e., $\mathbf{x}_r^{(k+1)} = \mathbf{W}_{rel}^{(k)} \mathbf{x}_r^{(k)}$ where $\mathbf{W}_{rel}^{(k)}$ is the learnable weight matrix.

- **KBGAT** [79] proposes attention-based aggregation process by considering both the entity embedding and relation embedding:

$$\mathbf{x}_h^{(k+1)} = g\left(\sum_{(r,t)\in\mathcal{N}_h} \alpha_{h,r,t}^{(k)} c_{h,r,t}^{(k)} \right) \tag{B.3}$$

where $c_{h,r,t}^{(k)} = \mathbf{W}_1^{(k)}[\mathbf{x}_h^{(k)} || \mathbf{x}_t^{(k)} || \mathbf{x}_r]$, $||$ is the concatenation operation. Note that the relation embedding is randomly initilized and shared by all layers, i.e, $x_r^{(k)} = x_r$. The coefficient $\alpha_{h,r,t}^{(k)}$ is the attention score for (h, r, t) in the k-th layer, which is formulated as follows:

$$\alpha_{h,r,t}^{(k)} = \frac{\exp(\mathrm{LR}(\mathbf{W}_2^{(k)} c_{h,r,t}^{(k)}))}{\sum_{(r,t')\in\mathcal{N}_h} \exp(\mathrm{LR}(\mathbf{W}_2^{(k)} c_{h,r,t'}^{(k)}))} \tag{B.4}$$

where LR is the LeakyReLU function, $\mathbf{W}_1^{(k)} \in \mathbb{R}^{d_{k+1}\times 3d_k}$, $\mathbf{W}_2^{(k)} \in \mathbb{R}^{1\times d_{k+1}}$ are two sets of learnable parameters.

For GNN-based models with K layers, we use $\mathbf{x}_h^{(K)}$, $\mathbf{x}_r^{(K)}$, and $\mathbf{x}_t^{(K)}$ as the final embeddings and denote them as $\mathbf{x}_h$, $\mathbf{x}_r$, and $\mathbf{x}_t$ for the simplicity of notations. Note that RGCN does not involve x_r in the aggregation component, which will be randomly initialized if required by the scoring function.

Table B.2 KGC results (%) with random graph structure for message passing process. The MPNN-based models still can achieve comparable performance.

		FB15K-237				WN18RR				NELL-995			
		MRR	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10
CompGCN	Original	35.5±0.1	26.4±0.1	39.0±0.2	53.6±0.3	47.2±0.2	43.7±0.3	48.5±0.3	54.0±0.0	38.1±0.4	30.4± 0.5	42.2±0.3	52.9± 0.1
	Random	35.3±0.1	26.3±0.1	38.7±0.1	53.4 ±0.2	47.3±0.0	44.0±0.2	48.5±0.2	53.8±0.3	38.8±0.1	31.1±0.0	42.8±0.1	53.3±0.1
RGCN	Original	29.6±0.3	19.1±0.5	34.0± 0.2	50.1±0.2	43.0±0.2	38.6±0.3	45.0±0.1	50.8±0.3	27.8±0.2	19.9±0.2	31.4±0.0	43.0±0.3
	Random	28.6±0.5	18.8±0.5	32.4±0.8	48.2±0.7	43.0±0.3	38.7±0.1	45.0±0.5	50.8±0.6	27.7±0.2	19.6±0.2	31.4±0.3	43.3±0.2
KBGAT	Original	35.0±0.3	26.0±0.3	38.5± 0.3	53.1± 0.3	46.4±0.2	42.6±0.2	47.9±0.3	53.9±0.2	37.4±0.6	29.7±0.7	41.4± 0.8	52.0±0.4
	Random	35.6±0.1	26.5±0.1	39.0±0.2	53.7±0.1	46.8±0.2	43.2±0.5	48.1±0.1	53.8±0.1	38.2±0.3	30.6±0.3	42.1±0.4	52.8±0.2

B.4 Scoring Function

Two widely used scoring function are DistMult [131] and ConvE [21]. The definitions of these scoring functions are as follows.

$$f^{DistMult}(h, r, t) = \mathbf{x}_h \mathbf{R}_r \mathbf{x}_t \tag{B.5}$$

$$f^{ConvE}(h, r, t) = g(\text{vec}(g([\overline{\mathbf{x}_h} || \overline{\mathbf{x}_r}] * \omega))\mathbf{W})\mathbf{x}_t \tag{B.6}$$

$\mathbf{R}_r \in \mathbb{R}^{d_k \times d_k}$ in Eq. (B.5) is a diagonal matrix corresponding to the relation r. In Eq. (B.6), $\overline{\mathbf{x}_h}$ denotes a 2D-reshaping of $\mathbf{x}_h$, ω is the convolutional filter, and $\mathbf{W}$ is the learnable matrix. vec($\cdot$) is an operator to reshape a tensor into a vector. || is the concatenation operator. ConvE feeds the stacked 2D-reshaped head entity embedding and relation embedding into convolution layers. It is then reshaped back into a vector that multiplies the tail embedding to generate a score.

For DistMult, there are different ways to define the diagonal matrix $\mathbf{R}_r$: For example, in RGCN, the diagonal matrix is randomly initialized for each relation r, while CompGCN defines the diagonal matrix by diagonalizing the relation embedding $\mathbf{x}_r$.

B.5 Loss Function

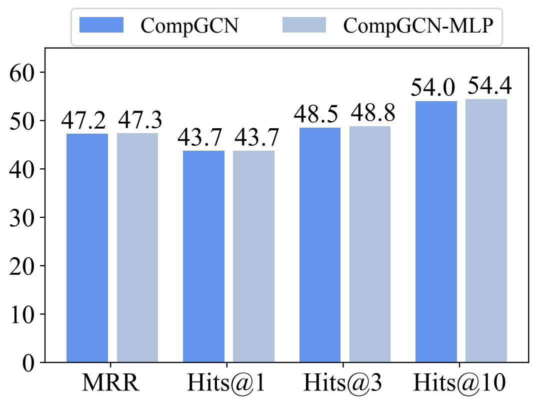
We adopt the Binary cross-entropy (BCE) as the loss function, which can be modeled as follows:

$$\mathcal{L} = -\sum_{(e_1, rel, e_2) \in \mathcal{D}^*_{train}} \Bigg(\log \sigma(f(e_1, rel, e_2)) + \sum_{(e_1, rel, e_2') \in C_{(e_1, rel, e_2)}} \log(1 - \sigma(f(e_1, rel, e_2'))) \Bigg). \tag{B.7}$$

where $f(\cdot)$ is the scoring function defined in the appendix B.4, and σ is the sigmoid function.

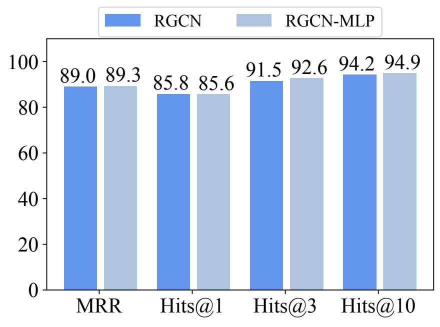
B.6 Does Message Passing Really Help KGC?

In section 3.4.1, We replace the message passing with the MLP while keeping other components untouched in CompGCN, RGCN and KBGAT. Due to the space limit, we only present the resutls on the `FB15K-237`dataset in section 3.4.1. In this section, we include additional results on other datasets. Specifically, we include results of CompGCN/CompGCN-MLP on the `WN18RR`dataset, RGCN/RGCN-MLP on the `WN18`and `FB15K`dataset, KBGAT/KBGAT-MLP on the `WN18RR`and `NELL-995`in Figure B.1, Figure B.2 and Figure B.3 respectively. All the counterpart MLP-based

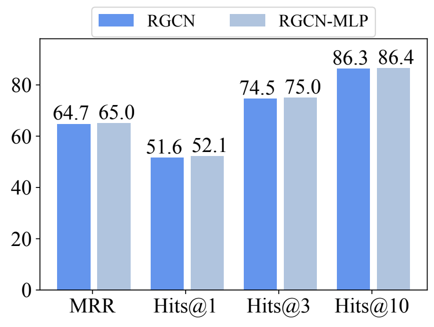


(a) WN18RR

Figure B.1 KGC results of CompGCN and CompGCN-MLP on WN18RR. On this dataset, CompGCN-MLP achieves compare performance as CompGCN.



(a) WN18



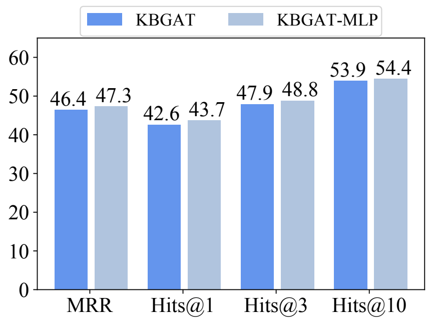
(b) FB15K

Figure B.2 KGC results of RGCN and RGCN-MLP on FB15Kand WN18. On all three datasets, RGCN-MLP achieves comparable performance as RGCN.

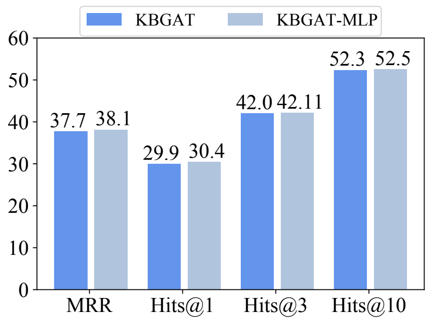
models achieve similar performance with the corresponding MPNN-based models, which show similar observations with the FB15K−237datasets in section 3.4.1.

B.7 MPNNs with random graph structure for message passing

In this section, we investigate how the performance perform when we use random generated graph structure in the message passing process. The number of random edges is the same as the ones in the original graph. When training the model by optimizing the loss function, we still use the original graph structure, i.e., $\mathcal{D}^*_{train}$ in Eq. (B.7) is fixed in Appendix B.5. Note that if the message passing has some contribution to the performance, aggregating the random edges should lead to the performance drop.



(a) WN18RR



(b) NELL-995

Figure B.3 KGC results of KBGAT and KBGAT-MLP on WN18RRand NELL-995. On all three datasets, KBGAT-MLP achieves comparable performance as KBGAT.

We present the results of CompGCN, RGCN and KBGAT on various datasets in Table B.2. We use “Original", “Random" to denote the performance with the original graph structure and random edges respectively. From Table B.2, we observe that using the noise edges achieves comparable performance, which further indicates that the message passing component is not the key part.

B.8 Time Complexity

We first define the sizes of weight matrices and embeddings of a single layer. We denote the dimension of entity and relation embeddings as d. The weight matrices in RGCN and CompGCN (shown in Eqs. (B.1) and (B.2) respectively in Appendix B.3.) are $d \times d$ matrices. In KBGAT (Eq. (B.3) in Appendix B.3), there are two weight matrices $\mathbf{W}_1$ and $\mathbf{W}_2$ of size $d \times 3d$ and $1 \times d$, respectively. Thus, the time complexity of RGCN, CompGCN, KBGAT for a single layer is $O(|E|d^2 + nd^2)$, $O(|E|d^2 + nd^2)$, $O(3|E|d^2 + nd^2 + |E|d)$, respectively, where $|E|$ is the number of edges and n is the number of nodes. While MLP doesn't have the message passing operation, the time complexity in a single layer is $O(nd^2)$. Note $|E|$ is usually much larger than n, thus the MLP is more efficient than MPNN.

B.9 Ensembling MLPs

We briefly introduce the MLP-based models we utilized for constructing the ensemble model in section 3.5.2 for the three datasets as follows:

- For the FB15K-237dataset, we ensemble the following models: DistMult + *w/o sampling*; DistMult + *with sampling* (two different settings with the number of negative samples as $0.5N$

and N, respectively); ConvE + *w/o sampling*; ConvE + *with sampling* (five different settings with the number of negative samples as $50, 200, 500, 0.5N$ and N, respectively).

- For the `WN18RR`dataset, we ensemble the following models: DistMult + *w/o sampling*; ConvE+ *w/o sampling*; ConvE+ *with sampling* (one setting with the number of negative samples as N).
- For the `NELL-995`dataset, we ensemble the following models: ConvE+ *w/o sampling*; ConvE+*with sampling* (five settings with the number of negative samples as $50, 200, 500, 0.5N$ and N).

APPENDIX C

LANGUAGE MODEL-ENHANCED LINK PREDICTION.

C.1 Implementation of Naive Fusion

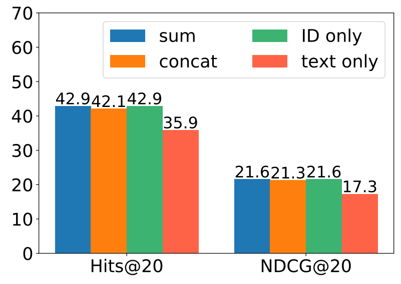
In this section, we give more details of the naive fusion methods in section 4.3.2.1: NFRec, UnisRec [38], and FDSA [148].

NFRec: It consists of several key components. We give more details of these components. **ID and text encoder**: We employ the same ID and text encoder as AlterRec which is introduced in section 4.4.1.1 and section 4.4.1.2, respectively. Through these two encoders, we obtain the item-level ID embedding $\mathbf{X}$ and text embeddings $\mathbf{H}$. **Fusion operation**: We fuse the ID and text item embeddings to form a final embedding $\mathbf{Z}$ via summation or concatenation to as mentioned in section 4.3.2.1. **Scoring function**: For a given session $\mathbf{s}$, we apply the mean function based on the fused item embedding to get the session embedding $\mathbf{q_s} = g(\mathbf{s}, \mathbf{Z})$, and then we use the vector multiplication between session embedding and the candidate item's fused embedding to get the score $y_{\mathbf{s},i} = \mathbf{Z_i}^T \mathbf{q_s}$. **Loss function**: We use the cross entropy as the loss function, which follows similar form with Eq. (4.3).

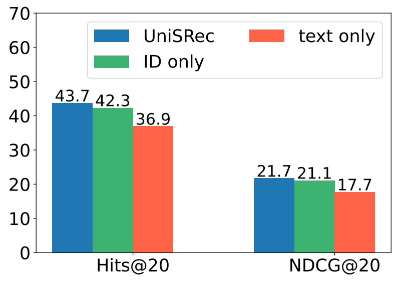
UniSRec [38]: The model employs the same ID encoder as NFRec, which utilizes a learnable embedding for the ID representation. For text encoder, it leverages a language model enhanced by the proposed adaptor to extract textual information. After pretraining the adaptor using two contrastive loss functions, it merges the ID and text embeddings through summation. UniSRec adopts the same cross-entropy loss function as used in NFRec. We use the official code of UnisRec [1] as the implementation.

FDSA [148]: The ID encoder generates ID embeddings using learnable embeddings. The text encoder employs a MLP and an attention mechanism to produce text embeddings. The Transformer is applied to items within a session to create ID and text session embeddings, which are then concatenated to form the final session embedding. Similar with NFRec and UniSRec, FDSA utilizes cross-entropy as the loss function. For the implementation of FDSA, we utilize code from the

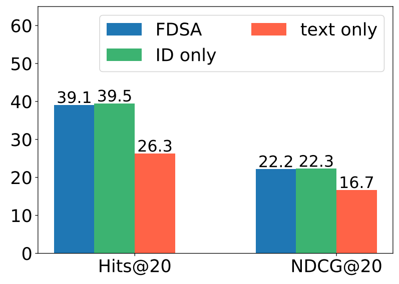
[1]https://github.com/RUCAIBox/UniSRec/tree/master



(a) NFRec

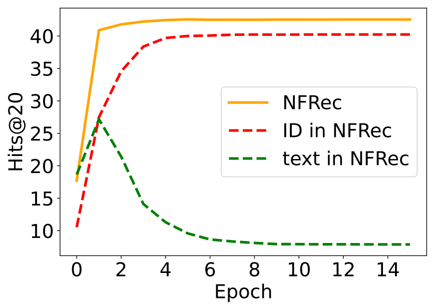


(b) UnisRec

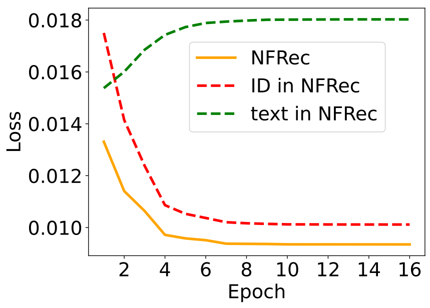


(c) FDSA

Figure C.1 Session recommendation results (%) on the HD. We compare the models combing ID and text against models trained independently on either ID or text information alone.



(a) HD



(b) HD

Figure C.2 Test performance in terms of Hits@20 (%) and training loss comparison on the HD dataset.

UniSRec's repository, which includes the implementation details for FDSA.

C.2 More Details when exploring NFRec

In this section, we give more details for the exploration conducted in section 4.3.2.2. We elucidate the process of dividing the NFRec into its ID and text components, and describe how we evaluate the performance and obtain the loss of "ID in NFRec" and "text in NFRec." Details on the implementation of the NFRec are provided in the Appendix C.1. For any given item i, we derive the ID embedding $\mathbf{X}_i$ and text embedding $\mathbf{H}_i$ from the corresponding ID and text encoders. These two embeddings are then concatenated to form a final embedding $\mathbf{Z}_i = [\mathbf{X}_i, \mathbf{H}_i]$. For a session $\mathbf{s} = \{s_1, s_2, ..., s_n\}$, the session embedding is obtained by applying the mean function to the final embeddings of the items within session $\mathbf{s}$: $\mathbf{q_s} = g_{mean}(\mathbf{s}, \mathbf{Z})$. This session embedding is represented as a concatenation of two parts derived from the ID and text embeddings, respectively:

$$\mathbf{q_s} = [\mathbf{q_s}^{ID}, \mathbf{q_s}^{text}] = [\frac{1}{|\mathbf{s}|}\sum_{s_i \in \mathbf{s}} \mathbf{X}_{s_i}, \frac{1}{|\mathbf{s}|}\sum_{s_i \in \mathbf{s}} \mathbf{H}_{s_i}]$$

The relevance score between a session and an item is then decomposed into two parts:

$$\begin{aligned} y_{s,i} = \mathbf{Z}_i^T \mathbf{q}_s &= [\mathbf{X}_i, \mathbf{H}_i]^T [\mathbf{q_s}^{ID}, \mathbf{q_s}^{text}] \\ &= \mathbf{X}_i^T \mathbf{q_s}^{ID} + \mathbf{H}_i^T \mathbf{q_s}^{text} = y_{s,i}^{ID} + y_{s,i}^{text} \end{aligned} \tag{C.1}$$

Thus, the relevance score in NFRec can be decomposed as the summation of the ID and text scores. Accordingly, we evaluate the performance and obtain the loss of "NFRec", "ID in NFRec" and "text in NFRec" based on $y_{s,i}$, $y_{s,i}^{ID}$ and $y_{s,i}^{text}$ in Eq. (C.1), respectively. For the loss function, the cross-entropy is employed.

C.3 Additional Results in Preliminary Study

Additional results on the HD dataset for investigations in sections 4.3.2.1 are displayed in Figure C.1 and Figure C.2, respectively. These figures indicate a trend similar to that observed with the Amazon-French dataset. Specifically, Figure C.1 reveals that models trained independently on ID data can achieve performance comparable to, or even surpassing, that of naive fusion methods. Furthermore, models relying solely on text information tend to perform the worst. In Figure C.2, it is observed that the ID component dominates the performance and loss. These findings are consistent with observations made with the Amazon-French dataset, suggesting that the phenomenon identified in observations 1 and 2 in section 4.3.2.1, as well as the imbalance issue in NFRec, may be prevalent across various datasets.