LARGE LANGUAGE MODEL-BASED RECOMMENDER SYSTEMS: INTEGRATING USER-ITEM FEATURES AND INTERACTION GRAPHS INTO LLMS

HUACHI ZHOU

PhD

The Hong Kong Polytechnic University

2026

The Hong Kong Polytechnic University
Department of Computing

Large Language Model-based Recommender Systems: Integrating User-Item Features and Interaction Graphs into LLMs

Huachi Zhou

A thesis submitted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy
December 2025

CERTIFICATE OF ORIGINALITY

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

Signature: _____

Name of Student: Huachi Zhou

Abstract

Large Language Models (LLMs) have recently been introduced into recommender systems to leverage their strong abilities in semantic understanding and reasoning beyond traditional models, raising the vision that the next generation of recommender systems could be LLM-based. However, simply plugging LLMs into recommendation pipelines, e.g., description-based prompting, raises three fundamental challenges: it is unclear whether LLMs truly learn the relationship between user–item features and interaction labels, how LLMs can directly exploit high-order structure in attributed graphs, and how to make such graph-integrated LLM recommenders efficient enough for real-world deployment. This thesis presents a systematic framework to address these challenges, advancing LLM-based recommendation by integrating user–item features into LLMs, extending this integration to attributed graphs, and ultimately enabling graph-based collaborative filtering (CF) on user–item interaction graphs with LLMs. First, we propose a self-monitoring framework that guides LLMs to align their predictions with informative user–item features. Next, we move to attributed graphs and introduce graph as a new language, enabling LLMs to capture high-order structure from new language corpus. Then we apply this idea to user–item interaction graphs and efficiently learn collaborative user/item embeddings from a sampled corpus for deployment. These three contributions form a coherent pipeline, demonstrating that with proper integration of user–item features and interaction graphs into LLMs, they can move beyond description-based prompting toward practical, scalable LLM-based recommender systems.

Publications Arising from the Thesis

1. Huachi Zhou, Kaijing Yu, Qinggang Zhang*, Hao Chen, Daochen Zha, Wenqi Pei, Anthony Kong, Xiao Huang, "Self-Monitoring Large Language Models for Click-Through Rate Prediction", in *ACM Transactions on Information Systems (TOIS)*, 2025.

2. Huachi Zhou†, Jiahe Du†, Chuang Zhou, Chang Yang, Yilin Xiao, Yuxuan Xie, Xiao Huang*. "Each Graph is a New Language: Graph Learning with LLMs", in *Findings of the Association for Computational Linguistics (ACL)*, 2025.

3. Huachi Zhou, Yujing Zhang, Hao Chen*, Qinggang Zhang, Qijie Shen, Feiran Huang, Xiao Huang, "LLM Collaborative Filtering: User-Item Graph as New Language", in *Association for the Advancement of Artificial Intelligence (AAAI)*, 2026.

Acknowledgments

I would like to begin by thanking my supervisor, Prof. Xiao Huang. He has been a steady source of support and clear guidance, while giving me enough freedom to follow my own ideas. His patience and practical advice, on research and on personal growth, have helped me make good decisions at many key moments. I have learned a lot from the way he frames difficult questions and breaks them down into doable steps, and that way of thinking has deeply influenced my work.

I am also grateful to my co-supervisor, Prof. Changwen Chen, Prof. Wei Ma, and Prof. Bo Li, for their instructive suggestions throughout this project. Their support and the resources they have made available are important in keeping the work on track and moving forward. I would like to thank several senior researchers whose input shaped this thesis in meaningful way. Prof. Hao Chen offered many helpful suggestions, especially on linking my work to real-world problems. I also appreciate the collaboration and insights from Prof. Qiaoyu Tan, Dr. Daochen Zha, Prof. Ninghao Liu, and Prof. Fan Yang; their comments and discussions have helped refine the thesis. My thanks also go to the members of the DEEP (Data Exploring and Extracting @ PolyU) Lab in the Department of Computing. Over the years, their feedback and encouragement make the hard parts easier and the good times better.

Finally, I want to thank my family and friends. They have supported me with understanding, celebrated small wins with me, and stood by me when things are difficult. This thesis would not have been possible without their trust and encouragement.

Table of Contents

Abstract i

Publications Arising from the Thesis ii

Acknowledgments iii

List of Figures ix

List of Tables xii

1 Introduction 1

1.1 Background 1

1.2 Motivation 2

1.3 Research Objectives 3

1.4 Contributions 4

1.5 Overall Structure 5

2 Literature Review 7

2.1 Input and Output of Recommender Systems 7

2.2 LLM-based CTR Prediction 7
2.3 LLM-based Attributed Graph Modeling 9
2.4 LLM-based CF 10

3 Self-Monitoring Large Language Models for Click-Through Rate Prediction 11

3.1 Introduction 11
3.2 Preliminaries 13
3.2.1 Problem Statement 13
3.2.2 Fine-Tuning with Prompt 14
3.3 Analysis of Fine-tuned LLMs 16
3.3.1 Feature Attribution Score Analysis 16
3.3.2 Performance on Head and Tail Items 19
3.4 Feature-Instructed Large language model for Monitoring (FILM) . . 21
3.4.1 Self-monitoring Temperature 21
3.4.2 Label Matching Loss 24
3.4.3 Transferring `FILM` to Lightweight Models 26
3.5 Experiments 27
3.5.1 Experimental Setting 28
3.5.2 Baselines 29
3.5.3 Implementation Details 31
3.5.4 Main Comparisons (RQ1) 33
3.5.5 Feature–interaction Relationships Analysis (RQ2) 33

3.5.6 Ablation Study (RQ3) 34
3.5.7 Backbone Generalization (RQ4) 35
3.5.8 Hyperparameter Sensitivity (RQ5) 36
3.5.9 Effectiveness of `FILM`-generated Embeddings (RQ6) 37
3.6 Summary 38

4 Each graph is a new language: Graph Learning with LLMs 40

4.1 Introduction 40
4.2 Methodology 42
4.2.1 Problem Setup and Learning Objective 42
4.2.2 Graph Language Pre-training 44
4.2.3 Structure-aware Fine-tuning 46
4.3 Discussion 48
4.3.1 Beyond Metaphor: The Statistical Isomorphism between Graph and Language 48
4.3.2 Theoretical Analysis: Information vs. Redundancy Trade-off . 49
4.4 Experiments 50
4.4.1 Datasets 50
4.4.2 Baselines and Variants 51
4.4.3 Implementation Details 53
4.4.4 Main Results (RQ1) 53
4.4.5 Ablation Study (RQ2) 54
4.4.6 Backbones and Hyperparameters (RQ3) 55

4.4.7 Efficiency Analysis (RQ4) 56
4.5 Summary 58

5 LLM Collaborative Filtering: User-Item Graph as New Language 59

5.1 Introduction 59
5.2 Preliminary 61
5.3 New Language Collaborative Filtering 62
5.3.1 Graph Corpus Collection 63
5.3.2 Collaborative Embedding Construction 70
5.4 NLCF Pseudocode 71
5.5 Experiments 73
5.5.1 Experimental Settings 73
5.5.2 Main Comparison (RQ1) 77
5.5.3 Efficiency Comparison (RQ1) 79
5.5.4 Ablation Study (RQ2) 79
5.5.5 Hyper-parameter Sensitivity (RQ3) 79
5.5.6 Online A/B Testing (RQ4) 80
5.6 Summary 81

6 Conclusion and Future Work 83

6.1 Conclusion 83
6.2 A Unifying Philosophy 84
6.2.1 From Generators to Representation Learners 84

6.2.2 Knowledge Injection for Domain Adaptation 85

6.2.3 Scalability and Deployment Readiness 85

6.3 Future Work 85

References **87**

List of Figures

1.1 The roadmap of this thesis. We progress from semantic alignment to structural learning, and finally achieve efficient recommendation. . . . 5

3.1 Attribution scores for Amazon Movies, Book-Crossing, and Amazon CDs. The first five models are traditional CTR methods that explicitly learn feature–interaction relations via crossing networks; the last is a fine-tuned base LLM. 15

3.2 The overall pipeline of the `FILM` framework. On the left, we show how user-item features are organized into the input prompt. In the middle, the prompt and the interaction embedding are used together to adjust the predicted interaction probability. On the right, samples with the same interaction label in a batch are pulled closer in the representation space to form more compact clusters. 20

3.3 Figure (a) shows the performance contribution from each component of our design. Figures (b) and (c) report `FILM`'s performance when we swap in Llama2-7B and Llama-3.1-8B as backbones, respectively. Figure (d) presents analogous results when replacing the backbone with TallRec and ClickPrompt. 32

3.4 Fine-tuned LLM shows only 1/4/1 positively attributed features across the three datasets, whereas `FILM` increases them to 2/4/3 respectively. 32

3.5 Attribution under item-ID perturbation on Amazon Movies. 35

3.6 t-SNE visualization of interaction embeddings from `FILM` without vs. with the label matching loss. 36

3.7 Hyperparameter sensitivity of `FILM` w.r.t. α and β on Amazon Movies, visualized as 3D surfaces for AUC and Logloss. Results for $\alpha = 10^{-4}$ are omitted due to training instability. 37

4.1 The figure demonstrates a comparison between mainstream methods and GDL4LLM for node-classification task. Figure (a) utilizes LLMs to embed node attributes and leverages GNN to aggregate the embeddings. Figure (b) presents the descriptions of graph structure centered around target nodes. Figure (c) illustrates how LLMs are pre-trained to capture graph structures through graph language learning, and how textual attributes are further integrated to enhance LLMs fine-tuning. 43

4.2 Test accuracy of GDL4LLM variants across five datasets. 54

4.3 Validation and test accuracy of GDL4LLM with Llama3-8B across five datasets. 55

4.4 Effect of sentence length l and number k on Wiki dataset. 56

5.1 Pipeline comparison between prior methods and NLCF. NLCF attains efficiency by training on short graph sentences and serving via precomputed user-item embeddings. 60

5.2 Overall NLCF pipeline. Top: collect corpus via random walks followed by similarity-based sampling. Bottom: fine-tune LLM on the corpus, then aggregate hidden states to build collaborative user/item embeddings for efficient retrieval. 63

5.3 Effect of backbone parameter scale on ML-10M dataset. 78

5.4 Comparison of sampling strategies on ML-1M dataset: random vs. similarity-based. 78

5.5 Impact of sentence length l and sampling ratio α on Precision@5 and NDCG@5 on ML-1M dataset. 81

List of Tables

3.1 Head/tail Logloss for traditional CTR baselines vs. fine-tuned LLMs. 18

3.2 Comparison with Traditional CTR Prediction Baselines. 29

3.3 Comparison with LLM-driven CTR Prediction Baselines. 30

3.4 Head/tail performance across CTR baselines, fine-tuned LLM, and `FILM`. 34

3.5 Performance under item-ID perturbation on Amazon Movies. OA/OL=Original AUC/Logloss; PA/PL=Perturbed AUC/Logloss. 35

3.6 Comparison of `FILM`-DeepFM with lightweight traditional CTR baselines. 38

4.1 Micro-F1 comparison across five datasets. 51

4.2 Dataset statistics. 52

4.3 Macro-F1 performance comparison across five datasets. 52

4.4 Token budget vs. modeled structural order (Token/(order)). 57

4.5 Training and inference time (hh:mm:ss). 57

5.1 Conceptual comparison between generation-based LLM CF and the proposed NLCF framework. 61

5.2 Precision with $N = 5$ and 10 on Steam, ML-10M, and ML-1M. 75

5.3 NDCG with $N = 5$ and 10 on Steam, ML-10M, and ML-1M. 77

5.4 Training and test time among LLM-based models (seconds [s], minutes [m], hours [h]). 80

5.5 Effect of similarity threshold t on NLCF performance (ML-1M). . . . 80

5.6 Online A/B tests on an industrial platform. 81

Chapter 1

Introduction

1.1 Background

Recommender systems have been a long-standing research topic. Despite their success in many real-world applications, traditional recommender models face two inherent limitations. (i) **Limited real-world knowledge.** Most traditional recommendation models are trained purely on user-item features and interactions. They have only narrow recommendation knowledge instead of general common sense [3, 81]. Therefore, they can only treat user-item features and interactions as abstract IDs or low-level numerical signals, without truly understanding the real semantic meaning behind. (ii) **Limited reasoning ability.** Traditional models are mainly designed to capture statistical correlations in user–item features and interactions [18]. They struggle to perform more complex reasoning over user–item features and interactions. The rapid development of large language models (LLMs), such as GPT and Llama [56, 26], has sparked strong interest in applying them to recommendation. These models have shown impressive capabilities in text understanding, reasoning, and generation [108], which naturally leads to the question: can LLMs serve as the core of the next generation of recommender systems? To integrate LLMs to recommender systems, a

common paradigm is to verbalize both types of information into natural language prompts: user and item features are expressed in text, and associated interactions are also described in language to model collaborative information [96]. Building on this prompt-based paradigm, recent studies have explored different ways of using LLMs for recommendation, which can be roughly grouped into two categories [47]: (i) The first category treats **LLMs as powerful in-context learners**. On top of directly asking the model for a final recommendation, these methods decompose the task into simpler sub-tasks and prompt the LLM to generate intermediate reasoning steps before producing the final answer [104]. At each step, the LLM may branch into multiple reasoning paths, reflect on previous mistakes, or revise earlier decisions [72]. In a similar spirit, some works further organize one or more LLMs into an agent framework [35], where multiple agents plan and manage task execution, maintain an external memory, and collaborate to complete complex recommendation tasks. (ii) The second category focuses on **fine-tuning LLMs** with parameter-efficient techniques [34], aligning them more directly with specific recommendation objectives. In this line of work, recommendation data are reorganized as instruction-style inputs, so that LLMs can learn to follow task-specific prompts and encode knowledge about target items and user preferences [97]. Depending on how strongly the LLM is involved, it may serve as a side-information encoder, e.g., providing semantic representations of user–item features or interaction [46], as the core recommender model itself [37], or be tightly coupled with traditional recommendation models in a hybrid architecture [39].

1.2 Motivation

Although LLM-based recommender systems are an appealing vision, directly applying LLMs to recommendation also raises several concerns. First, current LLMs are pre-trained on general text corpora and their objectives are not tailored for recommendation [2], so it is unclear whether they truly learn the relationship between

user–item features and interaction labels, rather than relying on a few salient tokens in the prompt. Second, the user–item interaction graph contains rich collaborative information, where high-order neighbors encode key user preferences [77]. However, LLMs are inherently designed for processing one-dimensional token sequences instead of graphs [10], and simply describing graph neighborhoods in natural language leads to long prompts and makes it hard for LLMs to capture high-order structural information. Finally, even if an LLM could encode high-order collaborative signals [61], LLM fine-tuning and generation-based inference over the entire item set would incur prohibitive latency and cost, which limits their practicality in real-world scenarios. To move beyond these limitations, we need an LLM that could comprehensively model user-item features in prediction and then could capture high-order structure information in interaction graphs and make them efficient in both training and inference for deployment, ultimately advancing the efficiency and effectiveness of LLM-based recommender systems.

1.3 Research Objectives

The central aim of this thesis is to move towards LLM-based recommender systems by improving how LLMs model users and items, from feature-level understanding to interaction-graph-level understanding. This goal is operationalized through three interconnected research objectives:

- **LLM-based User–Item Feature Integration.** The first objective is to investigate how LLMs use user-item features in recommender systems and to make this usage more comprehensive. This includes: (i) analyzing whether LLMs truly learn the relationship between textual features and interaction labels; and (ii) designing training objectives that encourage LLMs to boost their predictions on informative user-item features.

- **Attributed Graph-Aware LLMs.** The second objective is to enable LLMs to directly model graph structure, rather than relying only on textual attributes or long graph descriptions. Concretely, this involves: (i) devising a way to leverage the LLM's language-processing capabilities to learn graph structure in a natural manner; and (ii) studying whether training LLMs in this way allows them to capture high-order structure information in a token-efficient manner.
- **Efficient Interaction-Graph LLMs.** The third objective is to build an LLM-based framework that models the user–item interaction graph while remaining practical for real-world deployment. This includes: (i) exploring how to represent user–item interaction graphs as compact sequences that LLMs can efficiently learn from; (ii) understanding how high-order collaborative information can be encoded in these sentences; and (iii) designing a retrieval-friendly representation so that the learned information can be used without relying on expensive online LLM inference.

1.4 Contributions

This thesis proposes three contributions to LLM-based recommender systems, progressing from user–item feature integration to user–item graph integration.

- **Self-Monitoring LLMs for Click-through Rate (CTR) Prediction.** This thesis proposes a self-monitoring framework (FILM) that uses LLMs for CTR prediction while explicitly aligning predictions with user-item features. By adding simple self-monitoring and label-matching objectives, the model uses features more comprehensively and improves performance, especially on long-tail items, compared to simply fine-tuned LLMs.
- **Each Graph as a New Language for Graph-Aware LLMs.** We present a general method (GDL4LLM) that treats each graph as a new language, and

construct corpus from the graph. Training LLMs on these graph-defined corpus lets the model learn high-order graph structure without long natural language descriptions, and achieves strong results on node-level tasks.

- **Efficient LLM-based Collaborative Filtering (CF) on User–Item Interaction Graphs.** Building on the "graph as language" view, we develop an LLM-based CF framework (NLCF) that models the user–item interaction graph efficiently. We construct and sample a compact graph corpus to fine-tune LLMs, and then aggregate their hidden states into user/item embeddings for fast retrieval. This framework preserves high-order collaborative information while keeping both training and inference efficient.

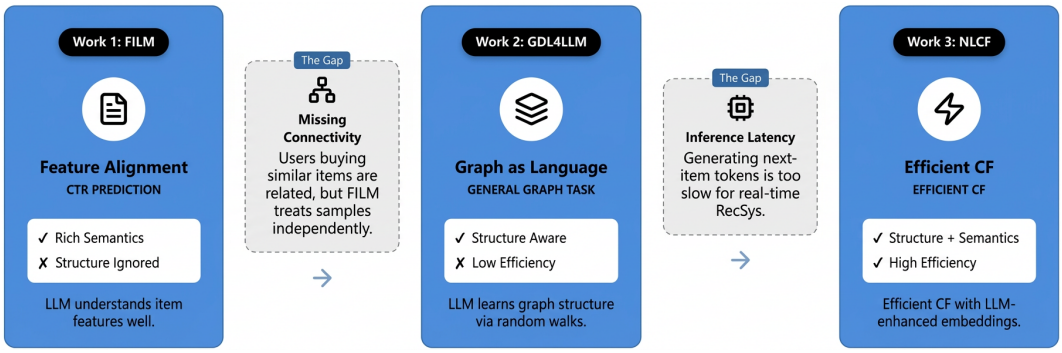


Figure 1.1: The roadmap of this thesis. We progress from semantic alignment to structural learning, and finally achieve efficient recommendation.

1.5 Overall Structure

This thesis is organized around a progression from integrating user–item features with LLMs to modeling user–item interaction graphs with LLMs as shown in Figure 1.1. After this introductory chapter, Chapter 2 gives background on LLM-based CTR prediction, LLM-based attributed graph modeling, and LLM-based CF, highlighting the limitations in these domains. Chapters 3-5 present the three core research papers.

Paper 1 [113] investigates how LLMs learn the relationship between user–item features and interaction labels for CTR prediction, and aims to better align LLM predictions with features. Paper 2 [111] views each attributed graph as a new language corpus and shows that LLMs trained on this corpus can effectively learn graph structure. Paper 3 [114] applies this idea to user–item interaction graphs and distills the sampled graph corpus into user/item embeddings for efficient online retrieval. The concluding chapter traces the overall trajectory of this thesis and points out several directions for future work.

Chapter 2

Literature Review

2.1 Input and Output of Recommender Systems

In recommender systems, there are two main types of input data. The first is user-item features, such as a user's living place and occupation and an item's category and title [48]. The second is user–item interactions, such as browsing and buying, which are usually organized as a user–item interaction graph where users and items are nodes and edges indicate observed interactions between them [6, 105]. The recommender system then outputs, for each user, a ranked list of candidate items, produced through a retrieval-and-ranking pipeline that first retrieves a small candidate set from the full item set and then scores and ranks these candidates [83]. In the pipeline, this thesis focuses on two representative ones: (i) *CTR prediction* and (ii) *CF*.

2.2 LLM-based CTR Prediction

CTR prediction aims to estimate the probability that a user will interact with an item based on user–item features [8, 92]. The essence of this task is to uncover meaningful

higher-order feature interactions within categorical or numerical fields [87] and capture rich relationships between input features and interaction labels. With the rapid development of LLMs, many recent studies use them for CTR prediction by converting user–item features into text and then either fine-tuning an LLM or prompting it directly to estimate interaction probabilities or rank items [94, 112]. Some works further enrich the prompt with short interaction histories or simple behavior statistics [95]. In this process, reinforcement learning or heuristics technique is used to sample which interactions to include [12, 76]. To leverage the strength of traditional CTR models in capturing higher-order feature interactions, a few methods also combine LLMs with conventional CTR predictors and train them jointly [39]. In industrial applications, modeling heterogeneous user behaviors with Large Language Models (LLMs) has become a prominent research direction, particularly involving multi-objective optimization tasks [103, 52]. A pioneering framework in this domain is OneRec [109], which was the first to implement an encoder-decoder architecture using semantic IDs for next-item prediction via reinforcement learning in a real-world scenario. Building on this, Minionerec [33] simplifies the architecture using an RQ-VAE tokenizer to better adapt to offline recommendation datasets, while GPR [98] improves generation accuracy by replacing the standard decoder with a dual heterogeneous hierarchical decoder. To facilitate large-scale evaluation, FORGE [11] releases a generative retrieval benchmark containing billions of SIDs, incorporating massive user behavior data and rich multimodal item features. Recent works have further refined SID representation and alignment: Onesearch [5] utilizes keyword-enhanced hierarchical quantization to emphasize multi-view user behaviors; DAS [90] bridges the gap between quantization and alignment stages through co-training to reduce information loss; and Align3GR [89] introduces multi-granularity alignment at the token, behavior, and preference levels. Additionally, OneLoc [74] extends these capabilities by integrating user location information into SID modeling and reward mechanisms. Despite these efforts, most existing studies focus on how to feed features into LLMs rather than how LLMs actually use these features. It remains unclear whether LLMs

truly learn a comprehensive mapping between user–item features and interaction labels, especially given that they are not originally designed for CTR prediction and may not naturally exploit all the provided information.

2.3 LLM-based Attributed Graph Modeling

The rise of LLMs has opened a new line of research on attributed graph understanding for tasks such as node classification and link prediction, especially when nodes or edges carry textual attributes, e.g, user-item interaction graph [60]. Since LLMs are strong at language understanding and reasoning, One line of work builds prompts that mix target node textual attributes with a small local subgraph centered around to obtain richer node and edge representations [36]. They could further enrich them with generated additional textual attributes or attributes aggregated from similar or neighboring nodes [82]. To enhance the performance, some methods also align these LLM-based representations with those from a teacher model, e.g., a well-trained graph encoder [57]. Another line mainly uses LLMs as graph augmentors: instruction-tuned LLMs refine edges to remove noise in the graph structure and a graph model is then trained on the updated structure [63, 49]. Based on this line, some works also generate labels for unlabeled nodes to support few-shot or even zero-shot learning on graphs [50, 51]. However, these methods face several issues. Using LLMs to directly reason over graphs is expensive, since two-dimensional structures must be linearized into long, detailed graph descriptions. Moreover, when LLMs are used only to augment graphs or provide labels, their capacity to directly learn and exploit graph structure is not fully utilized.

2.4 LLM-based CF

CF estimates how much a user will like an item by looking at users or items that have similar interactions, based on the idea that users with similar past interests are likely to prefer similar items in the future [19]. To bring LLMs into CF, one direct approach is to verbalize a user's interaction history as a prompt and feed it to an LLM for reasoning [30]. However, such purely text-based methods usually miss rich high-order collaborative information. A second line of work injects these information via precomputed embeddings [22, 30]: a conventional graph-based CF model is first trained on the full user–item graph to obtain user/item embeddings, and these embeddings are then consumed by the LLM, e.g., by replacing raw IDs in prompts [37, 101]. To better align these numerical embeddings with LLMs, some methods discretize embeddings into tokens [100] or even learn custom tokenizations and extend the vocabulary [30, 70, 58]. A third direction asks LLMs to generate side information, such as refined or external user/item descriptions, which are then fed into graph-based CF models [38, 27]. Across these lines, most approaches share one or two limitations: (i) they rely on expensive text generation or re-ranking over a small candidate set rather than efficient full-item set retrieval in the inference stage; and (ii) they do not leverage LLMs directly learn collaborative embeddings from the user–item interaction graph, but depend on precomputed representations, which can cause loss of collaborative information.

Chapter 3

Self-Monitoring Large Language Models for Click-Through Rate Prediction

3.1 Introduction

Having established the fundamental principles of CTR prediction task in Section 2.2, the effectiveness of a CTR prediction model is largely governed by two key capabilities: (i) deriving rich, informative representations from user–item features, and (ii) faithfully capturing how these features are associated with interaction labels [67, 84]. As discussed in Chapter 1, pre-trained LLMs are not explicitly trained for (ii): learning the mapping between user-item features and interaction labels, since such interaction records rarely appear in the pre-training data. After fine-tuning, an LLM can directly learn feature–interaction mappings from the CTR dataset. However, the combinatorial space of possible feature–interaction pairs is enormous and only sparsely instantiated in observed data [112]. Plain fine-tuning alone may still be insufficient to fully realize capability (ii). This motivates our main question: **to what extent**

do fine-tuned LLMs learn the feature–interaction relationships?

We investigate this question through complementary analyses. From a *feature-wise* perspective, we employ explainability tools to quantify how much user/item features contribute to the estimated probability in a fine-tuned LLM. We find lower feature attributions than those observed in advanced CTR methods, indicating that the LLM relies on a narrower subset of inputs. From an *interaction-wise* perspective, we compare performance on head versus tail items, where head items have abundant labels and tail items are underrepresented. Fine-tuned LLMs are competitive on head items but lag on tail items, suggesting difficulty in learning reliable feature–interaction relationships under label sparsity. Together, these results highlight room for improvement in capability (ii).

Two factors make this challenging. (i) **Inconsistent feature attention across predictions.** Trained primarily on general text, LLMs may shift attention across features for similar inputs, producing unstable feature prioritization. (ii) **Power-law data distribution.** Real-world datasets exhibit long tails; rare items and their associated feature combinations appear infrequently, making it difficult for fine-tuning to fully capture the mapping from those combinations to labels.

To address these issues, we propose Feature-Instructed Large language model for Monitoring (`FILM`), a simple yet effective framework that strengthens learning of feature–interaction relationships via two designs. First, we introduce a **self-monitoring temperature**. This module assigns a sample-specific, adaptive temperature in the loss function, encouraging the model to align its predictions more closely with the encoded feature information in the representation space. Second, we devise an **auxiliary compaction loss** that encourages compact separation between interaction-inducing and non-interaction-inducing feature patterns, so that interaction and non-interaction samples are more clearly separated in the representation space. With these two designs, `FILM` significantly outperforms state-of-the-art CTR baselines, with especially strong gains on tail items. We also show that embeddings produced by `FILM` can be

transferred to lightweight traditional CTR models, offering a practical deployment solution and further improving downstream performance.

Our contributions are summarized as follows:

- We perform feature-wise and interaction-wise studies to examine how well a fine-tuned base LLM captures feature–interaction relationships. Our results show that the LLM uses user-item features in a limited way and performs poorly on tail items with few interaction labels.
- We propose `FILM`, which integrates two simple mechanisms. Feature-wise, the self-monitoring temperature guides the LLM's attention toward informative user-item features. Interaction-wise, the auxiliary compaction loss helps the model better learn feature–interaction relationships for tail items.
- Experiments on three real-world datasets demonstrate that `FILM` outperforms strong CTR baselines, especially on tail items. Further analysis shows that `FILM` improves the learning of feature–interaction relationships and generalizes across different LLM backbones.

3.2 Preliminaries

3.2.1 Problem Statement

We consider a CTR prediction corpus $\mathcal{D} = (U, I, R)$. The set $U = \{U_1, \ldots, U_{|U|}\}$ contains user profiles, where each U_u aggregates attributes, e.g., age and location. The set $I = \{I_1, \ldots, I_{|I|}\}$ contains item profiles, where each I_i includes title, description, price, sales rank, etc. The binary interaction matrix $R \in \mathbb{R}^{|U|\times|I|}$ stores labels $r_{ui} \in \{0, 1\}$, indicating whether user u interacts with item i.

Given a pair (u, i), the goal is to estimate r_{ui} from the associated features. We cast

this as an instruction-style generation task: user/item features are verbalized into a prompt and tokenized as $\{x_1, \ldots, x_L\}$. The LLM is trained to predict the label token $x_{\text{label}} \in \{\text{“Yes”}, \text{“No”}\}$ for the final blank, aligning with the interaction label r_{ui}.

Prompt Templates. For each dataset, we design a template encoding user and item attributes and include user/item IDs for personalization [94]. We use:

Amazon Movies: *Given the user's and item's attributes, identify whether the user will like the target movie by answering Yes or No. Here is the information of the user [user_id]. Here is the information of the movie [movie_id]: Its title is [title], and its price is [price]. Its description is: [description]. Its sales rank among all movies is [sales rank]. Response: [ ]*

Book-Crossing: *Given the user's and item's attributes, identify whether the user will like the target item by answering Yes or No. Here is the information of the user [user_id]: The user is [age] years old and lives in [location]. Here is the information of the book [item_id]: The title is [title] written by [author] and published in [year of publication]. The publisher is [publisher]. Response: [ ]*

Amazon CDs: *Given the user's and item's attributes, identify whether the user will like the target item by answering Yes or No. Here is the information of the user [user_id]. Here is the information of the item [item_id]: Its title is [title], and its price is [price]. Its description is [description]. Its sales rank in item is [sales rank]. Response: [ ]*

We fill placeholders with features $\mathcal{U}_u, \mathcal{I}_i$; the LLM outputs “Yes” or “No” at the final position following [13].

3.2.2 Fine-Tuning with Prompt

We adopt parameter-efficient fine-tuning, letting the LLM map the prompt to a probability for each label token.

Interaction Probability Calculation

At the top transformer layer, the LLM produces token embeddings $\{e_1, \ldots, e_L\}$. A linear head with weights w_{Yes} and w_{No} projects the last-token embedding e_L to logits $l_{\text{Yes}}, l_{\text{No}}$, which are normalized by softmax to obtain $p_{\text{Yes}}, p_{\text{No}}$.

Fine-Tuning with LoRA

We apply Low-Rank Adaptation (LoRA) to reduce trainable parameters. For a base weight $\mathbf{W}_0 \in \mathbb{R}^{d \times d'}$, LoRA adds a low-rank update $\Delta\mathbf{W} = BA$ with $B \in \mathbb{R}^{d \times r}, A \in \mathbb{R}^{r \times d'}$, where $r \ll \min(d, d')$. We initialize A from a Gaussian and set $B = 0$, so $\Delta\mathbf{W} = 0$ at start. Only $\Delta\mathbf{W}$ is optimized; $\mathbf{W}_0$ remains frozen.

Figure 3.1: Attribution scores for Amazon Movies, Book-Crossing, and Amazon CDs. The first five models are traditional CTR methods that explicitly learn feature–interaction relations via crossing networks; the last is a fine-tuned base LLM.

3.3 Analysis of Fine-tuned LLMs

We study two aspects: (i) how strongly input features contribute to predictions (Section 3.3.1); and (ii) performance on head vs. tail items (Section 3.3.2).

3.3.1 Feature Attribution Score Analysis

To address question (i), we quantitatively analyze how much each input feature contributes to the predicted interaction label.

We adopt integrated gradients (IG) [62] to compute these attributions. Since the features are encoded as a sequence of tokens, we treat the tokens themselves as the units to which we assign contributions. Concretely, we measure how each input token affects the predicted probability p_{label} by looking at the gradients with respect to its embedding. Let $g(\cdot)$ be the attribution function and let the bold symbol $\mathbf{x}_i$ denote the embedding of input token x_i. The attribution score for token embedding $\mathbf{x}_i$ is defined as:

$$g(\mathbf{x}_i) = (\mathbf{x}_i - \mathbf{x}_{\text{base}}) \cdot \sum_{q=1}^{m} \frac{\partial p_{\text{label}}(\mathbf{x}_{\text{base}} + \frac{q}{m}(\mathbf{x}_i - \mathbf{x}_{\text{base}}))}{\partial \mathbf{x}_i} \cdot \frac{1}{m}. \tag{3.1}$$

Here, $\mathbf{x}_{base}$ is the starting point of the integration, which is typically set to the zero vector. We then build a straight-line path from $\mathbf{x}_{base}$ to the input token embedding $\mathbf{x}_i$ and divide this path into m equal steps. At the q-th step, we compute the gradient of the target prediction p_{label} with respect to the intermediate input $\mathbf{x}_q$, where $\mathbf{x}_q$ is given by the linear interpolation between $\mathbf{x}_{base}$ and $\mathbf{x}_i$.

Since gradients describe how small changes in the input affect the model output, Eq. (3.1) sums these gradients along the entire interpolation path from the baseline to the actual input. This path integral measures how each token gradually shifts the model's prediction from the baseline output to the final output. The below theoretical results show that this procedure gives a valid and consistent way to attribute the

prediction to the input tokens.

Theorem 1. *The sum of token attributions equals the change in the target probability:* $\sum_{i=1}^{L} g(\mathbf{x}_i) = p_{label}(\{x_i\}_{i=1}^{L}) - p_{label}(\{x_{base}\}_{i=1}^{L})$.

Proof. To simplify the notation, we represent $p_{label}(\{x_i\}_{i=1}^{L})$ and $p_{label}(\{x_{base}\}_{i=1}^{L})$ as $P(x)$ and $P(\bar{x})$ respectively. Let $\gamma(\alpha), \alpha \in [0,1]$ represent a parametric curve connecting x and $\bar{x}$, where $\gamma(0) = \bar{x}, \gamma(1) = x$, then we have

$$P(x) - P(\bar{x}) = P(\gamma(1)) - P(\gamma(0)), \tag{3.2}$$

$$= \int_0^1 \frac{dP(\gamma(\alpha))}{d\alpha} d\alpha, \tag{3.3}$$

$$= \int_0^1 \langle \nabla_\gamma P(\gamma(\alpha)), \gamma'(\alpha) \rangle d\alpha, \tag{3.4}$$

$$= \sum_i^L \int_0^1 [\nabla_\gamma P(\gamma(\alpha))]_i [\gamma'(\alpha)]_i d\alpha, \tag{3.5}$$

And we set $\gamma(\alpha)$ as the straight line between two points:

$$\gamma(\alpha) = (1-\alpha)x + \alpha\bar{x}, \tag{3.6}$$

Then the difference of predictions becomes:

$$P(x) - P(\hat{x}) = \sum_{i=1}^{L} |[\int_0^1 \nabla_\gamma P(\gamma(\alpha))|_{\gamma(\alpha)=(1-\alpha)x+\alpha\hat{x}} \, d\alpha]_i * [\hat{x} - x]_i|, \tag{3.7}$$

$$= \sum_{i=1}^{L} |[\frac{1}{m} \sum_{q=1}^{m} (\nabla_\gamma P(\gamma(\alpha)))|_{\gamma(\alpha)=(1-\alpha)x+\alpha\hat{x}, \alpha=\frac{q}{m}}]_i * [\hat{x} - x]_i|, \tag{3.8}$$

$$= \sum_{i=1}^{L} g(x_i). \tag{3.9}$$

□

The key idea of this theorem is that integrated gradients break down the predicted probability of the target label into a weighted sum of gradient integrals for the input tokens over a chosen interval. In this view, a positive attribution score for a feature

Table 3.1: Head/tail Logloss for traditional CTR baselines vs. fine-tuned LLMs.

Model	**Amazon Movies**		**Book-Crossing**		**Amazon CDs**	
	Head Logloss	Tail Logloss	Head Logloss	Tail Logloss	Head Logloss	Tail Logloss
DeepFM	0.2941	0.3001	0.4336	0.4813	0.2030	<u>0.2212</u>
DCN	0.2943	<u>0.2925</u>	<u>0.4326</u>	0.4802	**0.1964**	0.2239
FinalMLP	0.3013	0.2962	0.4336	**0.4769**	0.2124	0.2216
GDCNP	0.2973	0.3006	0.4359	<u>0.4780</u>	0.2026	0.2218
GDCNS	<u>0.2937</u>	**0.2906**	0.4379	0.4828	0.2024	**0.2210**
Base LLM	**0.2932**	0.3277	**0.4321**	0.4850	<u>0.1994</u>	0.2295

means that this feature pushes the model toward the correct target label. When more features receive positive attribution scores, the LLM can rely on a richer set of signals to recognize the target label. Using Eq.(3.1), we first compute an attribution score for every token in each test sample. We then aggregate these token-level scores within the same feature, across all samples in the test set, to obtain a feature-level score. In other words, for each feature, we sum its token scores over all test instances to derive its overall contribution for each model.

Applying IG to LLM attribution analysis presents certain limitations. First, *tokenization sensitivity*. Since LLMs process a single word as several tokens, the attribution score of a single semantic feature is often distributed across multiple tokens. To mitigate this, our analysis operates at the feature, where we sum the attribution scores of all constituent tokens belonging to a specific feature. This aggregation provides a unified estimate of the feature's total contribution to some extent. Second, *baseline selection*. IG requires a baseline input and we choose an all-zero vector. The physical meaning of the all-zero embedding is ambiguous, which may introduce bias, although it is a widely accepted default choice in the literature.

Attribution Score Visualization. We fine-tune LLaMA-7B using the strategy

in Section 3.2.2 on three real-world datasets: Amazon Movies, Book-Crossing, and Amazon CDs. For comparison, we also train several representative traditional CTR prediction models following the settings in Section 3.5.1. Figure 3.1 shows the resulting attribution scores.

Our analysis shows that input features do not provide strong positive contributions to the predictions of the fine-tuned LLM. In many cases, the LLM's decisions are only weakly supported by most features. For instance, on the Amazon Movies and Amazon CDs datasets, the predictions mainly rely on the item title and sales rank, while other features contribute very little. By contrast, traditional CTR models assign positive contributions to almost all features when predicting interaction labels on these two datasets, and their overall feature scores are consistently higher than those of the fine-tuned LLM.

These results highlight the strength of traditional CTR models, which explicitly construct feature combinations through feature-crossing networks and thus can better capture complex relations between features and interaction labels from training data. LLMs, however, are originally built for understanding and generating natural language, not for exhaustively modeling feature–interaction relationships. Simple fine-tuning may therefore be insufficient for them to fully learn such comprehensive relationships. This also suggests that LLM-based CTR models can still improve by involving more features more deeply in the prediction process and by learning richer feature–interaction patterns.

3.3.2 Performance on Head and Tail Items

To study question (ii), we split items into head and tail groups and pay special attention to model performance on tail items. Tail items have very few training examples, so predicting their interaction probabilities depends heavily on how well the model learns feature–interaction relationships. Following prior work [9, 110], we

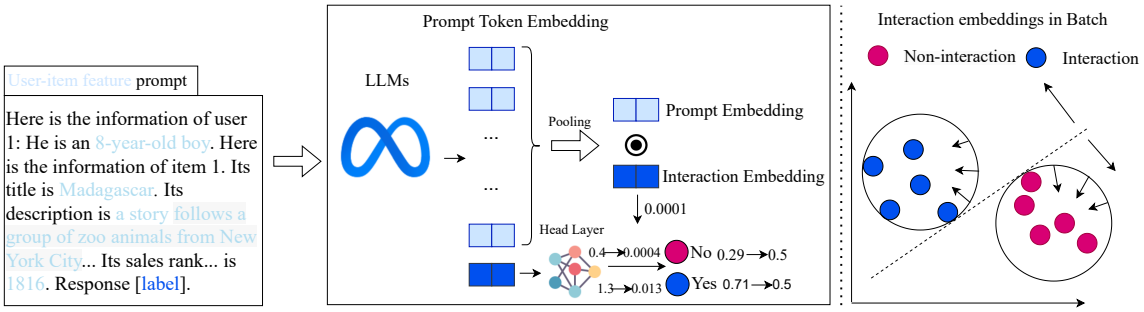


Figure 3.2: The overall pipeline of the `FILM` framework. On the left, we show how user-item features are organized into the input prompt. In the middle, the prompt and the interaction embedding are used together to adjust the predicted interaction probability. On the right, samples with the same interaction label in a batch are pulled closer in the representation space to form more compact clusters.

treat the top 20% of items with the most training samples as head items, and the rest as tail items.

We then compute the Logloss for each group, as reported in Table 3.1. The results show that fine-tuned LLMs perform worse than traditional CTR models on tail items, while achieving similar performance on head items. A natural explanation is that, for head items, the rich training data is enough for the LLM to adapt and fit interaction probabilities through fine-tuning. In contrast, tail items appear only a few times, making it hard for the LLM to estimate interaction probabilities reliably.

If the LLM is able to fully capture feature–interaction relationships, we would expect its performance on tail items to be much closer to that on head items. The clear gap we observe instead suggests that, under data scarcity, the LLM has difficulty learning these relationships, which directly harms its predictions on tail items.

3.4 Feature-Instructed Large language model for Monitoring (FILM)

In this section, we introduce `FILM`, which incorporates two plug-and-play designs, as illustrated in Figure 3.2. Feature-wise, a self-monitoring temperature penalizes low-confidence predictions with a larger training loss (Section 3.4.1). Interaction-wise, a label matching loss is introduced to compact the representation space of interaction and non-interaction labels, helping LLMs effectively differentiate between interaction-inducing and non-interaction-inducing features (Section 3.4.2). Finally, we discuss how to transfer `FILM` ranking capabilities to lightweight traditional CTR prediction models by using `FILM`-generated embeddings.

3.4.1 Self-monitoring Temperature

The core idea is to treat the correlation between feature and interaction embeddings as a confidence score, and use it to adapt both the interaction probability p_{Yes} and the interaction loss $\mathcal{L}_{\text{label}}$. This encourages the LLM to ground its predictions in relevant features. Prompt token embeddings encode contextualized feature information; weak correlation between a sample's interaction embedding and prompt embeddings indicates poor modeling of the feature–interaction relationship and should yield low confidence.

To address this, we introduce a self-monitoring mechanism that maps this correlation to a learnable temperature, which calibrates the predicted interaction probability during both training and testing. In training, low feature–interaction correlation yields a small temperature that pushes p_{Yes} toward 0.5. As shown in Theorem 3, this increases the interaction loss while shrinking gradients on interaction scores, so unreliable samples do not dominate learning. At test time, the temperature down-weights the interaction probability for weakly correlated samples, helping rank items

with better feature modeling higher and aligning predictions with feature quality.

To compute the temperature, we first obtain the interaction and prompt embeddings. We take the last prompt token embedding $\mathbf{e}_L$ as the interaction embedding, and derive the prompt embedding via mean pooling and a linear projection:

$$\mathbf{e}_c = \mathbf{W}_c \cdot \frac{1}{L} \sum_{j=1}^{L} \mathbf{e}_j,$$

where $\mathbf{W}_c$ is a learnable matrix. We then apply cosine similarity $s(\cdot)$ to get an initial temperature. To avoid trivial correlations, we add a discrimination step: we compare the similarity between $\mathbf{e}_L$ and its own prompt embedding $\mathbf{e}_c$ with similarities computed using other prompt embeddings $\mathbf{e}'_c$ and interaction embeddings $\hat{\mathbf{e}}_L$ within the same batch $\mathbb{B}$. The final self-monitoring temperature T for a sample is:

$$T = \frac{s(e_c, e_L)}{\sum_{(e'_c, \hat{e}_L) \in \mathbb{B}} s(e'_c, \hat{e}_L)}. \tag{3.10}$$

The reason that we use the normalized form instead of an alternative, e.g., simple Sigmoid, is twofold: (i) Relative calibration: we aim to measure how well a sample is aligned relative to the current batch statistics, rather than relying on an absolute threshold which is hard to tune. (ii) Training stability: in the early stages of fine-tuning, absolute cosine similarities might be uniformly low. A non-normalized T would be too small, causing vanishing gradients. The normalization ensures that even in early stages, samples contribute meaningful gradients. We then scale logits inside softmax:

$$p_{\text{Yes}} = \frac{\exp(l_{\text{Yes}} \cdot T)}{\exp(l_{\text{Yes}} \cdot T) + \exp(l_{\text{No}} \cdot T)}. \tag{3.11}$$

We multiply by T rather than divide by it, so small temperatures do not cause numerical issues such as NaN values. Then we train with cross-entropy:

$$\mathcal{L}_{\text{label}} = -(y \log p_{\text{Yes}} + (1 - y) \log(1 - p_{\text{Yes}})), \tag{3.12}$$

where $y = 1$ if x_{label} ="Yes" and $y = 0$ if x_{label} ="No". The following theorem explains how this design affects optimization:

Theorem 2. *The gradient of the LLM interaction score satisfies:* $\frac{\partial \mathcal{L}_{label}}{\partial l_i} \propto T$.

Proof. The derivative $\partial \mathcal{L}_{label} / \partial l_i$ of the loss towards the interaction score l_i is:

$$\begin{aligned}
\frac{\partial \mathcal{L}_{label}}{\partial l_i} &= -\sum_{j=1}^{2} y_j \frac{\partial \log(p_j)}{\partial l_i} = -\sum_{j=1}^{2} \frac{y_j}{p_j} \frac{\partial p_j}{\partial l_i} \\
&= -\frac{y_i}{p_i} \frac{\partial p_i}{\partial l_i} - \sum_{j \neq i}^{2} \frac{y_j}{p_j} \frac{\partial p_j}{\partial l_i} \\
&= -T \frac{y_i}{p_i} p_i (1 - p_i) - T \sum_{j \neq i}^{2} \frac{y_j}{p_j} (-p_j p_i) \\
&= -T y_i + T y_i p_i + T \sum_{j \neq i}^{2} y_j p_i = -T y_i + T p_i \sum_{j=1}^{2} y_j \\
&= T(p_i - y_i) \propto T.
\end{aligned} \tag{3.13}$$

This theorem shows that the gradient of the interaction loss w.r.t. interaction scores is scaled by the temperature. When the cosine similarity between prompt and interaction embeddings is low, the temperature – and thus the gradient magnitude – decreases, reducing the impact of samples with weak feature–interaction correlation and encouraging the model to rely on a broader feature set instead of overfitting to a narrow subset.

We set the temperature in the click-prediction loss via the cosine similarity between prompt and interaction embeddings for two reasons. First, LLMs are typically overconfident: they assign very high probability to the top prediction, almost none to alternatives, and lack calibrated confidence for unseen or rare feature combinations. Second, cosine similarity offers a cheap proxy for prediction confidence: higher similarity indicates stronger alignment between input features and the model's output representation [91], while lower similarity indicates less reliable predictions that should be down-weighted. Our temperature mechanism therefore provides a simple and efficient way to incorporate uncertainty estimation into LLM-based CTR prediction.

3.4.2 Label Matching Loss

Tail items, which have only a few interaction labels, make it hard for the model to learn and emphasize feature modeling. Their associated user–item features rarely appear in prompts, so the LLM does not see enough examples to form a thorough understanding of feature–interaction relationships. As a result, its predictions for these items can be unreliable. To strengthen the link between user–item features and interaction labels, especially for items with limited labels, we introduce a label matching loss that makes the representation boundary between interaction and non-interaction labels more separable.

The label matching loss is designed as an additional training objective. It encourages a compact representation space for interaction embeddings of both interaction and non-interaction labels. By pulling together interaction embeddings of tail items and other samples with the same label within the batch, the LLM can learn a clearer decision boundary and better align user–item features with their corresponding interaction labels.

To compact these interaction embeddings in the representation space, we first derive a closed-form expression for the volume occupied by a batch of interaction embeddings, and then explicitly shrink this volume.

Theorem 3. *Assume the rows in the interaction embedding matrix* $\mathbf{E}$ *have zero mean values. The volume of the space Vol(*$\mathbf{E}$*) satisfies the following:* $Vol(\mathbf{E}) \propto \log\det\left(\mathbf{I} + \frac{d}{|\mathbb{B}|\alpha^2}\mathbf{E}^\top\mathbf{E}\right)$,

Proof. We apply the SVD decomposition to the matrix $\mathbf{E} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$. The orthogonal matrices $\mathbf{U}, \mathbf{V}^\top$ represents the rotation and reflections of the matrix $\mathbf{E}$ while the singular values $\sigma_1, \sigma_2..., \sigma_d$ control the magnitude of orthogonal directions. And we use the eigenvalues $\lambda_1, \lambda_2..., \lambda_j$ of the covariance matrix $\mathbf{\Sigma}' = \mathbf{E}\left[\frac{1}{|\mathbb{B}|}\sum_{j=1}^{|\mathbb{B}|}\mathbf{e}_j\mathbf{e}_j^\top\right] = \frac{1}{|\mathbb{B}|}\mathbf{E}\mathbf{E}^\top \in \mathbb{R}^{d\times d}$ to replace the singular values:

$$Vol(\mathbf{E}) \propto \prod_{j=1}^{d} \sigma_j = \sqrt{\prod_{j=1}^{d} \lambda_j} = \sqrt{\det(\frac{1}{|\mathbb{B}|}\mathbf{E}\mathbf{E}^\top)}. \tag{3.14}$$

Since there may be zero eigenvalues, we add the scaled identity matrix to slightly perturb the original matrix. But the added matrix also takes up space, we use the space volume of the added matrix as the unit volume to estimate the volume of $\mathbf{E}$. $\hat{\mathbf{\Sigma}} = \mathbf{\Sigma}' + \frac{\varepsilon^2}{d}\mathbf{I} = \frac{\varepsilon^2}{d}\mathbf{I} + \frac{1}{|\mathbb{B}|}\mathbf{E}\mathbf{E}^\top \in \mathbb{R}^{d\times d}$, such that the following equation holds:

$$Vol\left(\mathbf{E}\right) \propto \frac{Vol\left(\mathbf{E}\right)}{Vol\left(Unit\right)} \propto \sqrt{\frac{\det\left(\frac{\varepsilon^2}{d}\mathbf{I} + \frac{1}{|\mathbb{B}|}\mathbf{E}\mathbf{E}^\top\right)}{\det\left(\frac{\varepsilon^2}{d}\mathbf{I}\right)}} = \sqrt{\det\left(\mathbf{I} + \frac{d}{|\mathbb{B}|\varepsilon^2}\mathbf{E}\mathbf{E}^\top\right)}. \tag{3.15}$$

□

In this theorem, $\mathbf{E} \in \mathbb{R}^{|\mathbb{B}|\times d}$ denotes the interaction embedding matrix for one mini-batch, where $|\mathbb{B}|$ is the batch size and d is the embedding dimension. The operator $\det(\cdot)$ denotes the determinant, $\mathbf{I}$ is the identity matrix, and ϵ is a tunable hyperparameter.

Based on this result, we next compute the volumes of the subspaces associated with interaction and non-interaction labels separately. Each interaction embedding in a batch can belong to either subspace with some probability. Embeddings with lower probability for a given label should have a smaller impact on the volume of that label's subspace. Therefore, we weight interaction embeddings by their predicted probabilities p_{Yes} and p_{No} when constructing the interaction and non-interaction spaces. Our goal is to sharpen the decision boundary by compressing both label-specific subspaces. To this end, we minimize the sum of their volumes. Let $\mathbf{E}_b$ denote the interaction embeddings in one batch. We define $\sum_{k=0}^{1} \frac{\mathrm{tr}(\mathbf{\Pi}_k)}{2|\mathbb{B}|} \cdot \log\det\left(\mathbf{I} + \frac{d}{\mathrm{tr}(\mathbf{\Pi}_k)\alpha^2}\mathbf{E}_b^\top\mathbf{\Pi}_k\mathbf{E}_b\right)$ where $\mathbf{\Pi}_k$ is a diagonal matrix whose i-th diagonal entry is the probability that the i-th sample belongs to the non-interaction label when $k = 0$, and to the interaction label when $k = 1$. By shrinking the volumes of these two label-conditioned spaces, the LLM can learn a clearer interaction decision boundary.

However, aggressively compressing both subspaces may gradually drive the entire representation space toward a single point. To avoid such collapse and preserve meaningful geometric structure, we also encourage the overall space spanned by all samples in the batch to remain sufficiently wide [93]. Combining these two terms yields the label matching loss L_{mat}, formally defined as:

$$L_{mat} = \sum_{k=0}^{1} \frac{\mathrm{tr}(\mathbf{\Pi}_k)}{2|\mathbb{B}|} \cdot \log\det\left(\mathbf{I} + \frac{d}{\mathrm{tr}(\mathbf{\Pi}_k)\alpha^2}\mathbf{E}_b^\top \mathbf{\Pi}_k \mathbf{E}_b\right) - \frac{1}{2}\log\det\left(\mathbf{I} + \frac{d}{|\mathbb{B}|\alpha^2}\mathbf{E}_b^\top \mathbf{E}_b\right). \tag{3.16}$$

The above loss has two merits: (i) Theoretical guarantee: Theorem 3 proves that optimizing the label matching loss is equivalent to compacting the representation space. (ii) No need for additional sample construction: CTR prediction datasets suffer from label imbalance [78], where one label may have a large number of samples while another has a few. Manually constructing more samples for the tail class may require multiple forward propagations, which is inefficient. Finally, we combine the interaction prediction loss and the label matching loss to fine-tune the LLM. The overall training objective is

$$\mathcal{L} = \mathcal{L}_{label} + \beta\mathcal{L}_{mat}, \tag{3.17}$$

where the hyperparameter β controls the strength of the label matching loss relative to the standard label prediction loss.

3.4.3 Transferring `FILM` to Lightweight Models

Although `FILM` achieves strong ranking performance, its inference cost is still high for latency-sensitive scenarios. To make the method practical for real-time CTR prediction, we introduce a resource-efficient strategy that transfers `FILM`'s ranking ability to lightweight, traditional CTR models, avoiding any LLM-related overhead at serving time. The key idea is to extract semantic embeddings from `FILM` and use them to enhance standard CTR models via knowledge transfer.

During FILM training, interaction embeddings encode rich dependencies between user–item features and interaction labels. And we use the interaction embeddings on the training set to learn compact user and item semantic embeddings, which can later be used as additional input features in lightweight CTR prediction models. We first obtain prompt-level representations from the trained FILM by averaging token embeddings within each prompt: $e_c = \frac{1}{L}\sum_{j=1}^{L} e_j$.

To construct semantically meaningful user and item representations, we adopt a dual-tower structure similar to DSSM [24]. This structure uses two separate multi-layer perceptrons (MLPs) to map the concatenation of the prompt embedding and interaction embedding into user-specific and item-specific semantic embeddings:

$$\mathbf{e}_u = \mathrm{MLP}([\mathbf{e}_c, \mathbf{e}_L]), \quad \mathbf{e}_i = \mathrm{MLP}([\mathbf{e}_c, \mathbf{e}_L]), \tag{3.18}$$

where $\mathbf{e}_u, \mathbf{e}_i \in \mathbb{R}^{\hat{d}}$ are the resulting user and item semantic embeddings, and $\hat{d}$ is chosen to match the embedding size used in downstream lightweight CTR models.

We train these MLPs with a reconstruction objective that encourages the semantic embeddings to preserve the interaction probabilities predicted by FILM. For each user–item pair (u, i) in a training batch $\mathbb{B}$, we minimize:

$$\mathcal{L}reco = \frac{1}{|\mathbb{B}|}\sum (u,i) \in \mathbb{B}(p_{Yes} - \mathbf{e}_u{}^{\top}\mathbf{e}_i)^2. \tag{3.19}$$

After training, the learned semantic embeddings can be plugged in as extra features for any lightweight CTR prediction model, such as DeepFM. This yields a favorable trade-off: the downstream model benefits from the high-level knowledge distilled from FILM, while its inference remains efficient, since it contains far fewer parameters than an LLM and can handle large-scale online traffic with low latency.

3.5 Experiments

Our experiments aim to answer the following research questions:

- **RQ1:** How does `FILM` perform compared with the state-of-the-art CTR prediction baselines?
- **RQ2:** How well does `FILM` learn feature-interaction relationships?
- **RQ3:** How does each design component contribute to `FILM`'s performance improvement?
- **RQ4:** How well does `FILM` generalize across different scenarios, including various LLM backbones, and LLM-driven CTR prediction backbones?
- **RQ5:** What is the impact of different hyper-parameters?
- **RQ6:** What is the effectiveness of the `FILM`-generated embeddings?

3.5.1 Experimental Setting

Data Preparation

We evaluate on three text-rich recommendation datasets: Amazon CDs [20], Amazon Movies [20], and Book-Crossing [116]. The two Amazon categories (CDs & Vinyl; Movies & TV) are drawn from a large-scale e-commerce platform and include abundant interactions and detailed item metadata.[1] The Book-Crossing corpus originates from an online book-sharing community and is publicly hosted on Kaggle.[2]

Following prior work [3], we binarize ratings into interaction labels. For Book-Crossing, ratings > 5 are mapped to positive ("Yes") and ratings < 5 to negative ("No"); for Amazon datasets, we use a threshold of 3. For Book-Crossing, we retain all available fields: *user ID, age, location, ISBN, title, author, publication year, publisher*. For Amazon datasets, many attributes are missing for a nontrivial portion of

[1] https://jmcauley.ucsd.edu/data/amazon/

[2] `https://www.kaggle.com/datasets/somnambwl/bookcrossing-dataset`

Table 3.2: Comparison with Traditional CTR Prediction Baselines.

Model	Amazon Movies		Book-Crossing		Amazon CDs	
	AUC	Logloss	AUC	Logloss	AUC	Logloss
DeepFM	0.8567	0.2969	0.7528	0.4514	0.8865	0.2136
DCN	0.8559	0.2938	0.7533	0.4504	0.8882	<u>0.2123</u>
FinalMLP	0.8547	0.2991	<u>0.7543</u>	<u>0.4497</u>	0.8827	0.2178
GDCNP	<u>0.8581</u>	0.2987	0.7513	0.4516	<u>0.8883</u>	0.2139
GDCNS	0.8580	<u>0.2929</u>	0.7530	0.4546	0.8867	0.2133
FILM	**0.8710**	**0.2823**	**0.7791**	**0.4436**	**0.9053**	**0.1985**

interactions; we therefore keep key attributes *user ID, item ID, title, price, description, sales rank*. Datasets are randomly split into train/validation/test with ratios 80% : 10% : 10% following [68].

Evaluation Metrics

In line with [64], we report AUC (Area Under ROC Curve) and Logloss (logistic loss). AUC measures the model's ability to rank positives above negatives, while Logloss captures the gap between predicted probabilities and binary labels.

3.5.2 Baselines

We compare against two groups of representative methods. (i) **Traditional CTR models** with explicit feature crossing:

- **DeepFM** [15]: combines FM [28] for linear crosses with a deep neural network (DNN) for nonlinear interactions.
- **DCN** [69]: augments a cross network with a low-rank parametrization and a

Table 3.3: Comparison with LLM-driven CTR Prediction Baselines.

Model	Amazon Movies		Book-Crossing		Amazon CDs	
	AUC	Logloss	AUC	Logloss	AUC	Logloss
BAIU	0.8560	0.2913	0.7527	0.4509	0.8832	0.2166
ClickPrompt	<u>0.8623</u>	0.2889	<u>0.7561</u>	0.4507	<u>0.8919</u>	<u>0.2062</u>
TallRec	0.8559	0.2962	0.7532	0.4529	0.8862	0.2385
P5	0.8465	0.3080	0.7492	0.4581	0.8858	0.2167
LlamaRec	0.8569	0.2975	0.7364	0.4954	0.8850	0.2238
KAC	0.8592	0.2939	0.7545	<u>0.4471</u>	0.8904	0.2095
w/o Loss-1	0.8379	**0.2638**	0.7387	0.4595	0.8049	0.3595
w/o Loss-2	0.5236	0.3202	0.7551	0.4569	0.5268	0.4163
FILM	**0.8710**	<u>0.2823</u>	**0.7791**	**0.4436**	**0.9053**	**0.1985**

deep tower.

- **GDCNS** [68]: uses a gated mechanism to filter unrelated high-order crosses.
- **GDCNP** [68]: presents a parallel stacking of linear and nonlinear parts.
- **FinalMLP** [54]: creates a two-stream MLP enhanced with feature gating and interaction aggregation.

(ii) **LLM-driven CTR baselines** that integrate LLMs into CTR prediction task:

- **BAIU** [85]: extracts semantic signals via LLM and feeds them to a classical CTR model.
- **TallRec** [3]: uses parameter-efficient fine-tuning to predict tokens in prompts.

- **P5** [13]: designs prompt formats for recommendation and label prediction.
- **LlamaRec** [95]: fuses a CTR model's score into the prompt.
- **ClickPrompt** [40]: jointly trains a soft-prompt generator with an LLM.
- **KAC** [81]: uses hybrid-adapter to convert LLM knowledge for CTR models.

(iii) **Variants of** `FILM` for ablation study analysis:

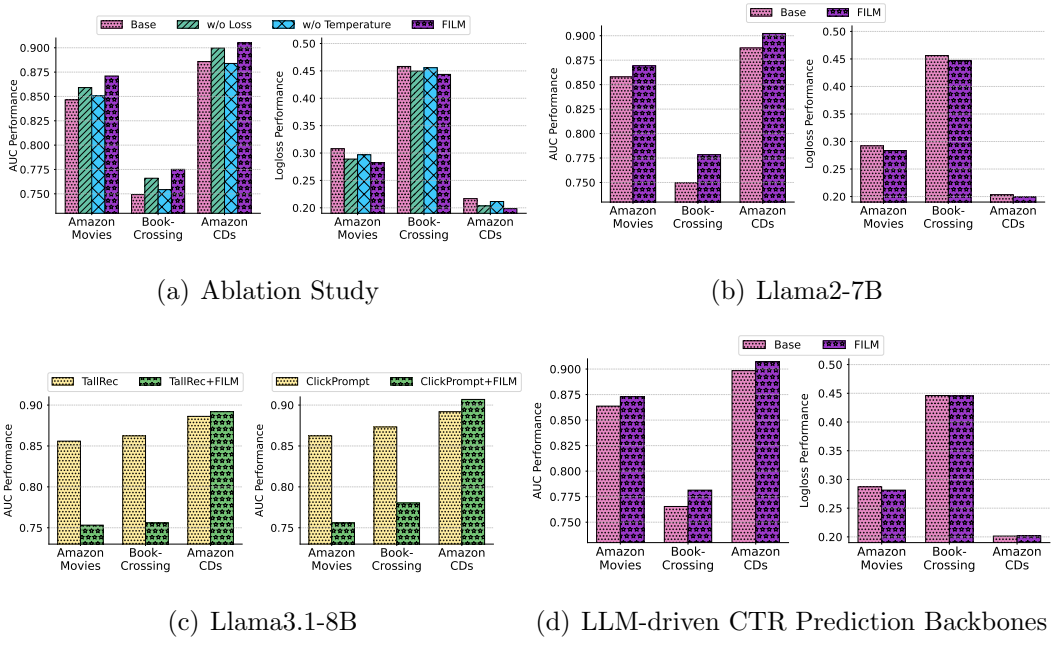
- **w/o Loss**: keep only the self-monitoring temperature.
- **w/o Temperature**: keep only the label matching loss.
- **w/o Loss-1**: remove the first term in the label matching loss.
- **w/o Loss-2**: remove the second term in the label matching loss.

3.5.3 Implementation Details

We use official public implementations for all baselines. LLaMA-7B [65] serves as the default LLM backbone; LLaMA-2-7B [65] and LLaMA-3.1-8B [1] are used for generalization. Training runs on NVIDIA Tesla A100 80GB. For LLM-based models, LoRA dropout, rank, and α are set to 0.2, 8, and 16; weight decay is 1e-2; optimizer is AdamW with lr=1e-4; batch size 32. Hyperparameters α, β are selected from $\{1e^0, 1e^{-1}, 1e^{-2}, 1e^{-3}, 1e^{-4}\}$. Each configuration is repeated 10 times.

For traditional CTR models, we tune DNN hidden sizes from $\{512, 256, 128, 64, 32\}$, L2 from $\{1e^0, 1e^{-1}, 1e^{-2}, 1e^{-3}, 1e^{-4}\}$, and dropout from $\{0, 0.05, 0.1, 0.15, 0.2, 0.25\}$, selecting by validation. Batch size is 1024.

For attribution analysis, we set IG steps $m = 50$. Because the LLM consumes the entire prompt, at step q we scale each input token embedding by $q/50$, obtain gradients for all tokens, and average across steps to obtain token-level attributions.



(a) Ablation Study

(b) Llama2-7B

(c) Llama3.1-8B

(d) LLM-driven CTR Prediction Backbones

Figure 3.3: Figure (a) shows the performance contribution from each component of our design. Figures (b) and (c) report `FILM`'s performance when we swap in Llama2-7B and Llama-3.1-8B as backbones, respectively. Figure (d) presents analogous results when replacing the backbone with TallRec and ClickPrompt.

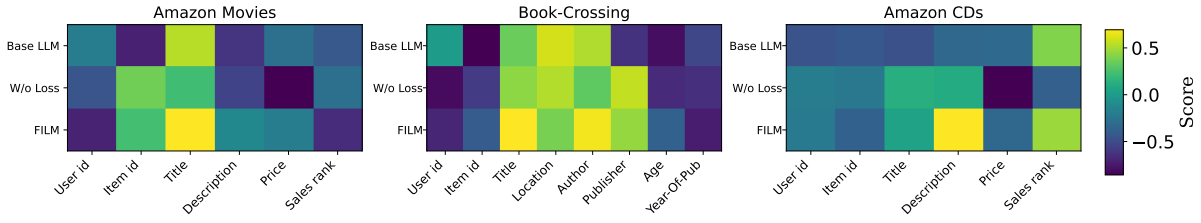


Figure 3.4: Fine-tuned LLM shows only 1/4/1 positively attributed features across the three datasets, whereas `FILM` increases them to 2/4/3 respectively.

3.5.4 Main Comparisons (RQ1)

Tables 3.2 and 3.3 summarize results across the three datasets. We highlight three findings. ① **Explicit feature crossing helps estimate interaction probabilities.** Within traditional baselines, deeper or gated cross networks, e.g., DCN, GDCN variants, FinalMLP generally outperform DeepFM, indicating that richer feature crossing captures more informative feature–interaction relationships. ② **Augmenting prompts with additional features benefits LLM-driven CTR.** Methods coupling LLMs with ID-based models, e.g., ClickPrompt, BAIU, outperform pure LLM fine-tuning, e.g., P5, suggesting complementarity: LLMs contribute semantics, while CTR models better learn feature–interaction relations, particularly for tail entities. Yet these hybrids still fall behind our approach. ③ `FILM` **achieves the best overall results.** `FILM` strengthens how fine-tuned LLMs capture feature–interaction relationships. Unlike prior LLM-based CTR predictors, it explicitly addresses the tendency of LLMs to ignore user-item features when ranking candidate items. By recalibrating feature modeling in the cross-entropy loss, `FILM` increases the model's sensitivity to feature signals in CTR prediction. In addition, it sharpens the interaction decision boundary by separately compacting the representations of interaction and non-interaction samples. Taken together, these findings indicate that `FILM` effectively unlocks additional performance from LLMs, enabling them to surpass traditional CTR prediction baselines.

3.5.5 Feature–interaction Relationships Analysis (RQ2)

We first inspect feature utilization via attribution visualizations in Figures 3.1 and 3.4, and then evaluate head/tail performance following [46] in Table 3.4. ④ `FILM` **increases positive feature usage and improves tail generalization.** The self-monitoring temperature encourages reliance on informative features, while the label matching loss compacts label-wise embedding spaces, yielding clearer boundaries. This explains the

Table 3.4: Head/tail performance across CTR baselines, fine-tuned LLM, and FILM.

Model	Amazon Movies		Book-Crossing		Amazon CDs	
	Head Logloss	Tail Logloss	Head Logloss	Tail Logloss	Head Logloss	Tail Logloss
DCN	0.2943	<u>0.2925</u>	<u>0.4326</u>	0.4802	<u>0.1964</u>	0.2239
FinalMLP	0.3013	0.2962	0.4336	<u>0.4769</u>	0.2124	<u>0.2216</u>
GDCNP	0.2973	0.3006	0.4359	0.4780	0.2026	0.2218
Base LLM	<u>0.2932</u>	0.3277	0.4321	0.4850	0.1994	0.2295
FILM	**0.2786**	**0.2884**	**0.4266**	**0.4722**	**0.1886**	**0.2056**

significant gains on tail items.

To evaluate attribution reliability, we perturb Amazon Movies by masking 75% of item-ID token embeddings. Table 3.5 and Figure 3.5 show that performance drops align with attribution dependence: the base LLM changes marginally due to low attribution, traditional CTR models degrade moderately, and FILM degrades the most, consistent with its higher reliance on item ID. Attributions reallocate from the masked feature to others while decreasing for item ID, supporting stability and faithfulness.

3.5.6 Ablation Study (RQ3)

To address the third research question, we conduct an ablation study that isolates each component of FILM and examines its individual effect. The corresponding results are reported in Figure 3.3. We summarize our findings as follows: ⑤ **every component in FILM contributes positively to performance.** The base model alone performs relatively poorly, mainly due to two issues: weak exploitation of feature information and a blurry interaction decision boundary, especially for tail items.

Both ablated variants, removing self-monitoring temperature or removing label match-

Table 3.5: Performance under item-ID perturbation on Amazon Movies. OA/OL=Original AUC/Logloss; PA/PL=Perturbed AUC/Logloss.

Model	OA	OL	PA	PL
DeepFM	0.8567	0.2969	0.8511	0.2971
GDCNP	0.8581	0.2987	0.8538	0.2974
GDCNS	0.8580	0.2929	0.8544	0.2931
Base	0.8465	0.3080	0.8452	0.3085
FILM	0.8710	0.2823	0.8634	0.2906

Figure 3.5: Attribution under item-ID perturbation on Amazon Movies.

ing loss, perform better than the base model, with the variant that retains label matching loss achieving the larger gain. This suggests that, without temperature calibration, the LLM does not sufficiently emphasize user-item features in the interaction embeddings. When we activate both components simultaneously, the performance improves substantially, indicating that they are complementary.

3.5.7 Backbone Generalization (RQ4)

To answer RQ4, we evaluate `FILM`'s generalizability across: (i) different LLM backbones and (ii) different LLM-driven CTR backbones on Figure 3.3. We observe that ⑥ **`FILM` consistently improves performance across backbones.** The gains on LLaMA-2-7B and LLaMA-3.1-8B suggest that even strong LLMs may underutilize features under sparse supervision; `FILM` encourages attention to informative user–item attributes and strengthens feature–interaction learning. Furthermore, `FILM` can be integrated with other LLM-driven CTR predictors to further unlock performance. ⑦ **We also find that performance is sensitive to model size**: larger LLM backbones consistently surpass their smaller counterparts, indicating that increased ca-

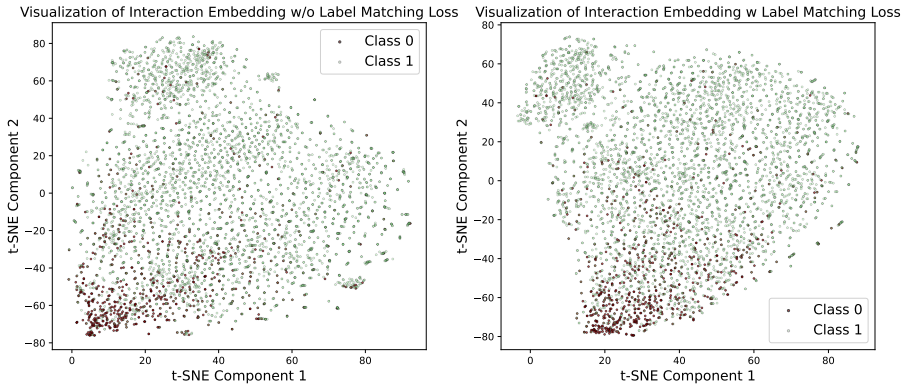
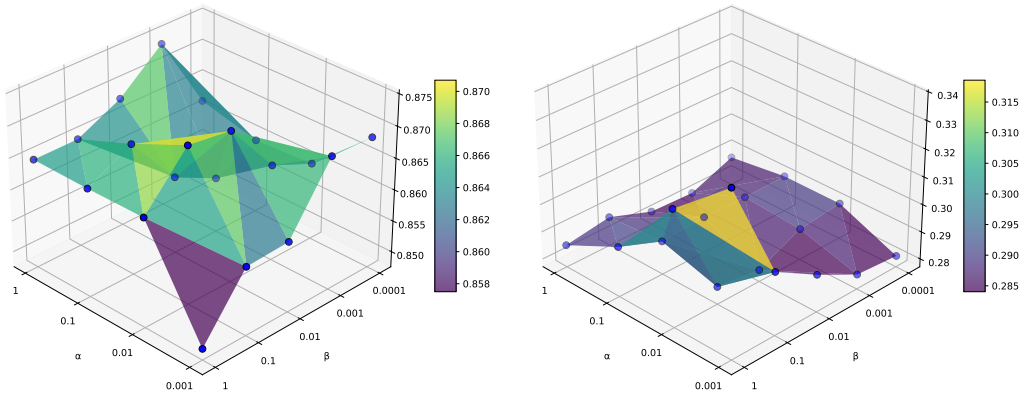


Figure 3.6: t-SNE visualization of interaction embeddings from `FILM` without vs. with the label matching loss.

pacity helps the model encode more complex feature–interaction relationships. This trend aligns with established scaling-law observations.

3.5.8 Hyperparameter Sensitivity (RQ5)

To examine how the hyperparameters α and β jointly affect performance, we sweep one while holding the other fixed. The resulting three-dimensional landscapes on the Amazon Movies dataset are presented in Figure 3.7(a) and Figure 3.7(b). The main observations are as follows: **⑧ Overall, moderate settings of α and β yield the best performance.** For α, increasing its value effectively enlarges the representation space for interaction and non-interaction labels. However, if α becomes too large, the representation clusters spread out excessively, making it difficult for the LLM to separate interaction labels and causing a noticeable performance drop. When α is too small, the opposite issue arises: interaction representations are over-compressed and may collapse toward a single point, which also severely harms performance. For β, small values weaken the influence of the label matching loss, thus limiting its ability to enforce a clear interaction decision boundary. On the other hand, overly large β places too much emphasis on this regularization term, disrupting normal interaction probability learning and leading to degraded results. Across all datasets, the trends



(a) Impact of α and β on AUC on Amazon Movies dataset

(b) Impact of α and β on Logloss on Amazon Movies dataset

Figure 3.7: Hyperparameter sensitivity of FILM w.r.t. α and β on Amazon Movies, visualized as 3D surfaces for AUC and Logloss. Results for $\alpha = 10^{-4}$ are omitted due to training instability.

are consistent: extreme values of α or β are suboptimal, while intermediate choices of both hyper-parameters achieve the highest AUC and lowest LogLoss.

3.5.9 Effectiveness of FILM-generated Embeddings (RQ6)

To assess the utility of the embeddings produced by FILM, we feed the reconstructed user and item semantic embeddings into a lightweight DeepFM-based CTR prediction model as additional feature inputs. The resulting model, denoted as FILM-DeepFM, is then benchmarked against a broad collection of practically deployable baselines, as reported in Table 3.6. We intentionally omit baselines that depend on LLM calls at inference time, since their reliance on expensive LLM querying contradicts our objective of efficient real-world deployment. The comparison shows that FILM-DeepFM consistently surpasses all conventional CTR prediction baselines across three datasets and evaluation metric. Even relative to KAC, which already exploits knowledge distilled from pre-trained LLMs, FILM-DeepFM achieves superior performance. We attribute

Table 3.6: Comparison of FILM-DeepFM with lightweight traditional CTR baselines.

Model	Amazon Movies		Book-Crossing		Amazon CDs	
	AUC	Logloss	AUC	Logloss	AUC	Logloss
DeepFM	0.8567	0.2969	0.7528	0.4514	0.8865	0.2136
DCN	0.8559	<u>0.2938</u>	0.7533	0.4504	0.8882	0.2123
FinalMLP	0.8547	0.2991	0.7543	0.4497	0.8827	0.2178
GDCNP	0.8581	0.2987	0.7513	0.4516	0.8883	0.2139
GDCNS	0.8580	0.2929	0.7530	0.4546	0.8867	0.2133
KAC	<u>0.8592</u>	0.2939	<u>0.7545</u>	<u>0.4471</u>	<u>0.8904</u>	<u>0.2095</u>
FILM-DeepFM	**0.8606**	**0.2908**	**0.7560**	**0.4462**	**0.8915**	**0.2090**

this advantage to two main reasons: (i) the learned embeddings are explicitly aligned with the CTR prediction task, in contrast to methods that simply reuse generic LLM representations without task-specific adaptation; and (ii) traditional CTR models such as DeepFM and DCN mainly depend on feature-cross networks to capture feature–interaction relationships, but lack the richer semantic modeling capabilities that the FILM embeddings introduce.

3.6 Summary

This chapter examines how fine-tuned LLMs internalize the relationship between user–item features and interaction labels in CTR prediction. Through both feature-wise and interaction-wise analyses, we reveal that base LLMs often neglect user–item features when estimating interaction probabilities, leading to particularly weak performance on tail items.

To mitigate these issues, we proposed `FILM`, a framework that augments LLM fine-tuning and inference with two simple but effective mechanisms. The first component, self-monitoring temperature, dynamically adjusts the prediction loss by tracking how well the model utilizes feature information, thereby encouraging more faithful feature modeling. The second component, label matching loss, explicitly compacts the representations of interaction and non-interaction labels to form a sharper decision boundary. Together, these designs drive LLMs toward comprehensive feature–interaction learning, yielding substantial performance improvements over both strong traditional baselines and recent LLM-driven predictors.

Limitations. Despite these advancements, a fundamental limitation remains: `FILM` operates under the assumption that interactions are independent and identically distributed. It treats each user–item interaction as an isolated instance based solely on textual semantics, thereby overlooking the high-order neighbors that naturally exists in recommendation scenarios. In real-world systems, users buying similar items are implicitly related, creating a rich web of collaborative information that feature-level modeling alone cannot capture.

This limitation motivates the transition to the next chapter, where we extend our scope from semantic feature space to structural graph space. Specifically, we propose to model the recommendation problem through the lens of the attributed graph. As we will discuss, this formulation serves as a unified prototype, capable of generalizing across interaction graphs, etc. By treating the attributed graph as a language, we aim to equip LLMs with the ability to reason over these structural dependencies, filling the gap between isolated semantics and collaborative information.

Chapter 4

Each graph is a new language: Graph Learning with LLMs

4.1 Introduction

As discussed in Section 2.3, many works try to bring LLMs into text-attributed graph learning following two directions: description-based and summarization-based approaches. Both approaches struggle with higher-order neighborhoods. Long free-form descriptions are redundant and poorly structured, and complex connections become messy when written as text, leading to long prompts and weak reasoning. On the other hand, approaches that mainly use attribute embeddings do not involve structure deeply in the LLM's training. Multi-order aggregation can cause over-smoothing and separates the LLM's optimization from structural learning, making it hard to learn high-order relations well.

As a result, it is hard to give LLMs a representation of high-order neighborhoods that is both compact and accurate. Unlike GNNs that build message passing into the model, LLMs usually need to linearize the graph into sequences, which increases context length and compute. Also, dependencies across different orders grow quickly

as neighborhoods expand, which makes concise representations difficult.

We take inspiration from cross-lingual transfer: LLMs trained mostly on one language can still work well on another after seeing a relatively small amount of data. Based on this, we view each graph as a new language and teach the LLM that language. Our framework, Graph-Defined Language for Large Language Models (GDL4LLM), replaces natural language descriptions with a graph language corpus built from the graph. We map nodes to special tokens and create graph sentences by random walks. Pre-training on these sentences helps the LLM learn structural patterns. We also show that this pre-training objective encodes structure, such as how node degree affects uncertainty, into the model. For downstream tasks, we sample short graph sentences centered on a target node to compactly capture multi-hop information, and we can attach textual attributes to provide semantic information.

During fine-tuning, we keep the structure knowledge learned in pre-training by freezing the pre-trained LoRA weights and adding a new set of LoRA weights for the specific task. This keeps the learned grammar of the graph language and allows efficient adaptation to node classification. The sampled graph sentences give a concise view of multi-hop structure without long prompts. If needed, we also concatenate the textual attributes of visited nodes so the LLM can use its strength in natural language understanding. In short, by treating a graph as a language and training LLMs to understand that language, GDL4LLM offers compact yet structure-aware representations that scale to higher-order dependencies, providing a simple and effective approach for learning on text-attributed graphs.

Our main contributions are:

- We treat graph structure learning for LLMs as learning a language and provide a theoretical argument that the pre-training objective stores structural properties in the model.
- We present GDL4LLM, which builds a graph language corpus with random

walks for pre-training and then fine-tunes with short, target-centered graph sentences, optionally combined with node textual attributes.

- On five real datasets, GDL4LLM outperforms description-based methods and attribute-embedding baselines. It achieves higher accuracy with fewer tokens and faster inference, and it captures higher-order neighborhoods effectively.

4.2 Methodology

This section presents GDL4LLM, a two-stage framework that treats a text-attributed graph as a learnable language. We first formalize the problem and training objective, then detail (i) the construction of a graph language corpus and pre-training of an LLM on that corpus, and (ii) a structure-aware fine-tuning procedure that preserves structural knowledge while adapting to node classification. Throughout, we integrate notation and task definitions that would otherwise appear in preliminaries.

4.2.1 Problem Setup and Learning Objective

Graph and attributes. Let a text-attributed graph be denoted by $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{X})$, where $\mathcal{V} = \{v_1, \ldots, v_{|\mathcal{V}|}\}$ is the node set, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the edge set, and $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_{|\mathcal{V}|}\}$ contains node-level textual attributes. The topology is encoded by an adjacency matrix $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ with $\mathbf{A}_{ij} = 1$ if $(v_i, v_j) \in \mathcal{E}$ and 0 otherwise. Each $\mathbf{x}_i$ may be a document such as a paper abstract or product description.

Node classification. Given $\mathcal{G}$, our goal is to produce a representation $\mathbf{t}_i \in \mathbb{R}^d$ for every v_i that fuses structure and text, and to predict a class label in $\{1, \ldots, C\}$. We

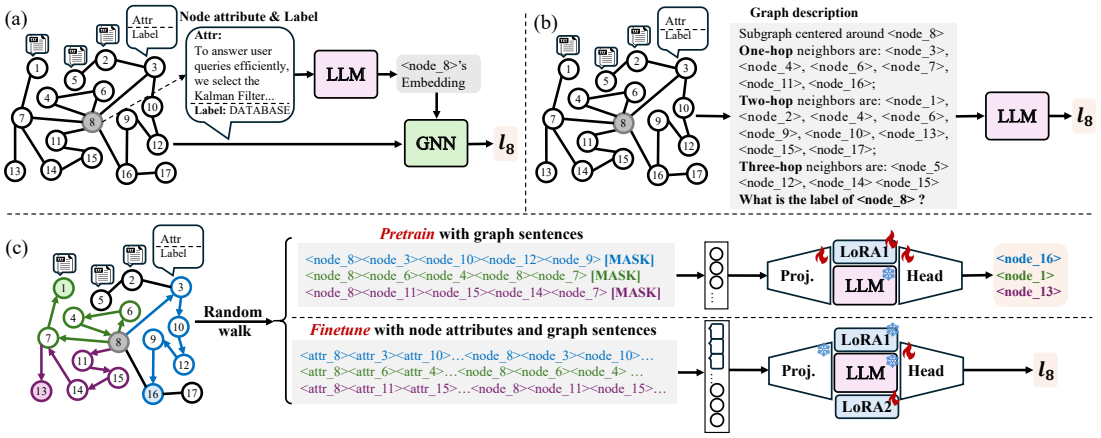


Figure 4.1: The figure demonstrates a comparison between mainstream methods and GDL4LLM for node-classification task. Figure (a) utilizes LLMs to embed node attributes and leverages GNN to aggregate the embeddings. Figure (b) presents the descriptions of graph structure centered around target nodes. Figure (c) illustrates how LLMs are pre-trained to capture graph structures through graph language learning, and how textual attributes are further integrated to enhance LLMs fine-tuning.

use a linear classifier with parameters $\hat{\mathbf{W}}_h \in \mathbb{R}^{d\times C}$ and bias $\mathbf{b} \in \mathbb{R}^C$:

$$\hat{y}_i^c = \frac{\exp\left(\hat{\mathbf{W}}_{h,c}^\top \mathbf{t}_i + b_c\right)}{\sum_{\tilde{c}=1}^{C} \exp\left(\hat{\mathbf{W}}_{h,\tilde{c}}^\top \mathbf{t}_i + b_{\tilde{c}}\right)}. \tag{4.1}$$

Training minimizes the cross-entropy over labeled nodes:

$$\mathcal{L}_{\text{CE}} = -\sum_{i=1}^{|\mathcal{V}|}\sum_{c=1}^{C} y_i^c \log \hat{y}_i^c, \tag{4.2}$$

where $y_i^c \in \{0,1\}$ is the ground-truth indicator.

Parameter-efficient tuning. We adopt Low-Rank Adaptation (LoRA) for efficient fine-tuning of a frozen LLM. For a base weight $\mathbf{W}_0 \in \mathbb{R}^{d\times d'}$ in the LLM, we introduce a trainable low-rank update $\Delta\mathbf{W} = BA$ with $B \in \mathbb{R}^{d\times r}$, $A \in \mathbb{R}^{r\times d'}$, and $r \ll \min(d, d')$. We initialize $B = \mathbf{0}$ and A with a Gaussian distribution so that $\Delta\mathbf{W}$ starts at zero and only the small factors are optimized while $\mathbf{W}_0$ remains fixed.

4.2.2 Graph Language Pre-training

We convert the graph into a discrete language over an extended vocabulary and pre-train an LLM to model token transitions induced by the topology. This stage equips the model with structural priors without resorting to verbose natural language descriptions.

Vocabulary and Syntax of the Graph Language

Tokens (nodes). We assign each node v_i a unique special token, e.g., `<node_i>`, by expanding the tokenizer's vocabulary. These tokens are out-of-vocabulary w.r.t. the base LLM and will be embedded via a learnable projector.

Sentences (paths). A *graph sentence* is a sequence of tokens following edges in $\mathcal{G}$; semantically, it encodes a walk in the graph. For example, a sentence may look like `<node_8><node_3><node_10><node_12><node_9>`. Collecting many such sequences yields a corpus $\mathcal{C} = \{s_1, s_2, \dots\}$ that reflects local and higher-order structures.

Sampling the Corpus via Random Walks

To expose the model to structure around every node, we perform k random walks of length l starting from each node token. Let $\mathcal{N}(v_i) = \{v_j \mid (v_i, v_j) \in \mathcal{E}\}$ denote the neighbors of v_i. The transition probability is

$$P(v_j \mid v_i) = \begin{cases} \frac{\mathbf{A}_{ij}}{\sum_{u \in \mathcal{N}(v_i)} \mathbf{A}_{iu}}, & \text{if } v_j \in \mathcal{N}(v_i), \\ 0, & \text{otherwise.} \end{cases} \tag{4.3}$$

This procedure produces approximately $k \cdot |\mathcal{V}|$ sentences, jointly covering 1-hop through l-hop neighborhoods, with token frequency governed by degrees and local edges.

Embedding New Tokens and Language Modeling Objective

Projecting node tokens. Because <node_i> tokens are newly introduced, we obtain initial semantic anchors by encoding textual attributes $\mathbf{x}_i$ with the LLM and then applying a learnable linear projector $\mathbf{W}_p \in \mathbb{R}^{d\times d}$ to align these anchors with structure. Intuitively, $\mathbf{W}_p$ learns to map nodes with similar local structure to nearby embeddings while preserving text-driven semantics.

Next-token pre-training. We endow the LLM with a trainable head $\mathbf{W}_h \in \mathbb{R}^{d\times|\mathcal{V}|}$ over node tokens and attach a LoRA adapter $\Delta\mathbf{W}_1$ to selected LLM layers during pre-training. Given a sentence $s_i = (s_{i,1}, \ldots, s_{i,l}) \in \mathcal{C}$ and hidden states $\{\mathbf{t}_q\}_{q=1}^{l}$, we maximize the likelihood of the next token:

$$\mathcal{L}_{\text{pre}} = -\sum_{q=1}^{l} \log P(s_{i,q} \mid s_{i,1:q-1};\, \mathbf{W}_p, \Delta\mathbf{W}_1, \mathbf{W}_h)\,, \tag{4.4}$$

where the distribution over node tokens at step q is parameterized by logits $\mathbf{W}_h^\top \mathbf{t}_q$.

What is learned structurally.

Theorem 4. *For a language model with sufficient capacity to construct the inner product between all $\mathbf{W}_{h,q}$ and $\mathbf{t}_q$ pairs, given node $s_{i,q}$ with degree d_q, then $\mathbf{W}_{h,q}\cdot\mathbf{t}_q \propto \log\left(\frac{\mathbb{I}_{(s_{i,q-1},s_{i,q})\in\mathcal{E}}\cdot\sum_{\mathbf{A}}}{d_q}\right)$.*

Proof. Assume that all possible token weight vectors $\mathbf{W}_{\text{h,q}}$ form a weight vector set $\mathcal{W}$, and all hidden representations of the next possible graph token $\mathbf{t}_q$ form a context set $\mathcal{T}$. We use $\#(\cdot,\cdot)$ to represent the number of co-occurrences of two quantities. If the language model has sufficient capacity to construct the inner product between any pair $\mathbf{W}_{\text{h,q}}$ and $\mathbf{t}_q$, the optimization of each pair's inner product does not affect the others. Then we replace the cross entropy loss in the pre-training process with the binary cross entropy loss. The overall pre-training loss is calculated as:

$$\mathcal{L} = \sum_{\mathbf{w}\in\mathcal{W}} \sum_{\mathbf{t}\in\mathcal{T}} \#(w,t) \left(\log \left(\frac{\exp(\mathbf{w}\cdot\mathbf{t})}{1+\exp(\mathbf{w}\cdot\mathbf{t})} \right) \right) + \sum_{\mathbf{w}\in\mathcal{W}} \sum_{\mathbf{t}\in\mathcal{T}} \#(w,t) \cdot |\mathcal{W}| \cdot \mathbb{E}_{t\sim P_t} \left[\log \left(\frac{1}{1+\exp(\mathbf{w}\cdot\mathbf{t})} \right) \right].$$

The pre-training loss with respect to a specific pair is calculated as:

$$\mathcal{L}_{(\mathbf{w},\mathbf{t})} = \#(\mathbf{w},\mathbf{t}) \log \left(\frac{\exp(\mathbf{w}\cdot\mathbf{t})}{1+\exp(\mathbf{w}\cdot\mathbf{t})} \right) + |\mathcal{W}| \cdot \#(\mathbf{w}) \cdot k \log \left(\frac{1}{1+\exp(\mathbf{w}\cdot\mathbf{t})} \right),$$

where k is a constant factor. We calculate the gradient of the above term and set it to zero, obtaining the following terms:

$$e^{2x} - \left(\frac{\#(\mathbf{w},\mathbf{t})}{|\mathcal{W}| \cdot \#(\mathbf{w}) \cdot d_q} - 1 \right) e^x = \frac{\#(\mathbf{w},\mathbf{t})}{|\mathcal{W}| \cdot \#(\mathbf{w}) \cdot k}. \tag{4.5}$$

The inner product $\mathbf{w}\cdot\mathbf{t}$, i.e., x, satisfies the following equation:

$$\mathbf{w}\cdot\mathbf{t} \propto \log \left(\frac{\mathbb{I}_{(s_{i,q-1},s_{i,q})\in\mathcal{E}} \cdot \sum_{\mathbf{A}}}{d_q d_{q-1}} \right). \tag{4.6}$$

□

The above theorem and proof shows how node degrees shape the pre-training objective. Nodes with higher degrees tend to have smaller optimal inner products, while nodes with lower degrees have larger ones, which matches the difference in certainty for sampled graph sentences. Through this training process, the LLM gradually picks up key structural signals, such as node degrees and how nodes are connected.

4.2.3 Structure-aware Fine-tuning

We now adapt the pre-trained graph language model to node classification while preserving structural knowledge and incorporating textual semantics.

Retaining Structural Knowledge with Disjoint Adapters

Let $\mathbf{W}_0$ be the frozen LLM weights and $\Delta\mathbf{W}_1$ the pre-trained structural adapter. For downstream tasks, we introduce a separate LoRA branch $\Delta\mathbf{W}_2$:

$$\mathbf{W} = \mathbf{W}_0 + \Delta\mathbf{W}_1 + \Delta\mathbf{W}_2, \tag{4.7}$$

and only optimize $\Delta\mathbf{W}_2$ while keeping $\mathbf{W}_0$ and $\Delta\mathbf{W}_1$ fixed. We reinitialize a task-specific classifier $\hat{\mathbf{W}}_h \in \mathbb{R}^{d\times C}$ and a new projector $\hat{\mathbf{W}}_p$, to better align text and graph structure for the target label space optimization..

Graph Language Conditioning for Classification

For a target node v_i, we sample a small set of graph sentences $\{s_1, \ldots, s_k\}$ of length l using (4.3), each starting from `<node_`i`>`. This mini-corpus compactly captures the subgraph structure. We condition the LLM on these sentences and minimize

$$\mathcal{L}_{\text{CE}} = -\log P\Big(y_i \,\Big|\, s_{1:k};\, \hat{\mathbf{W}}_p, \Delta\mathbf{W}_2, \hat{\mathbf{W}}_h\Big), \tag{4.8}$$

which couples the preserved structural prior via $\Delta\mathbf{W}_1$ with the task adapter $\Delta\mathbf{W}_2$. Choosing l and k balances local specificity and higher-order coverage without incurring long natural language prompts.

Injecting Textual Semantics

To simultaneously exploit structural and semantic signals, we integrate the graph language corpus with the textual attributes of nodes. The graph language encodes structure regularities, whereas the textual attributes supply complementary semantic context to the prompt. For each node, we construct a composite document by traversing its sequence in the graph language corpus and concatenating the textual attributes of all visited nodes. This composite document is paired with the graph

language during both pre-training and fine-tuning, allowing the LLM to better interpret inter-node relationships and to produce more accurate node classifications via its natural language reasoning abilities.

4.3 Discussion

4.3.1 Beyond Metaphor: The Statistical Isomorphism between Graph and Language

While the concept of "graph as language" serves as the foundational metaphor for GDL4LLM, it is grounded in specific statistical properties shared between attributed graph and natural language. We summarize these shared properties to delineate the validity scope of this metaphor.

Power-Law Distributions. The first connection lies in the statistical distribution of their constituent elements. Natural language vocabulary follows Zipf's law, where a few words appear frequently while the majority are rare. Similarly, most real-world graphs are scale-free networks where node degrees follow a power-law distribution .

Probabilistic Alignment. The core mechanism bridging the two domains is probabilistic dependency. In natural language, the semantic relationship between words is fundamentally captured by their co-occurrence probability within a sequence. Similarly, in graphs, the structural relationship between nodes is governed by the transition probability of moving from one node to another. Our random walk strategy acts as a translation layer that explicitly maps the graph's transition probabilities into token co-occurrence probabilities. This ensures that the LLM learns topological proximity using the exact same objective function it uses to learn linguistic grammar, validating the "graph as language" metaphor at a mathematical level.

4.3.2 Theoretical Analysis: Information vs. Redundancy Trade-off

While our corpus construction strategy is empirically justified, it involves a fundamental theoretical trade-off between maximizing structural information and minimizing token redundancy. We formulate this as an optimization problem from an information-theoretic perspective.

Let $\mathcal{C}_{k,l}$ be the constructed graph corpus, parameterized by the number of walks per node k and the walk length l. The quality of the corpus can be measured by the mutual information between the graph topology and the generated sequences, denoted as $I(\mathcal{G}; \mathcal{C}_{k,l})$.

Our objective is to maximize this mutual information subject to a computational budget, e.g., total graph token count $N \propto |V| \cdot k \cdot l$. This can be expressed as the following optimization objective:

$$\max_{k,l} \quad \mathcal{J}(k,l) = I(\mathcal{G}; \mathcal{C}_{k,l}) - \lambda \cdot \text{Cost}(k,l) \tag{4.9}$$

where λ is a Lagrange multiplier controlling the penalty for computational cost. The trade-off is shown in two key aspects:

- **Walk Length l:** Increasing l allows the corpus to capture higher-order proximities and long-range dependencies. However, as l goes longer, the random walk converges to the stationary distribution and subsequent tokens become statistically independent of the starting node, contributing noise rather than local structure information. This leads to diminishing returns in $I(\mathcal{G}; \mathcal{C}_{k,l})$.
- **Walk Number k:** Increasing k improves the estimation of local transition probabilities via the Law of Large Numbers. However, excessive sampling leads to redundancy, where the marginal information gain of adding a new sequence

approaches zero $\frac{\partial I(\mathcal{G};\mathcal{C}_{k,l})}{\partial N} < \epsilon$.

Therefore, our parameter choices for k and l aim to locate the "knee point" of this optimization landscape. We seek a corpus that is sufficiently informative to reconstruct the graph structure while remaining compact enough to avoid diluting the LLM's attention mechanism with redundant token transitions.

4.4 Experiments

We evaluate GDL4LLM on five public benchmarks to quantify accuracy, dissect the contribution of each component, study sensitivity to design choices, and measure efficiency. Concretely, we aim to answer: **RQ1** How does GDL4LLM perform on node classification compared with competitive text-attributed graph methods? **RQ2** What are the roles of graph language pre-training and attribute-aware prompting, and how much do they contribute individually and jointly? **RQ3** How do core hyperparameters and backbone LLMs influence performance? **RQ4** How efficiently does GDL4LLM capture higher-order structural context in terms of tokens and time cost?

4.4.1 Datasets

We consider five datasets: ACM, Wikipedia, Amazon, Ogbn-Arxiv, and Ogbn-Products. Table 4.2 summarizes basic statistics. Detailed dataset descriptions as well as the associated train/validation/test splits are provided. Following prior work [102], we partition the first three datasets into training, validation, and test sets with an 8:1:1 ratio. For the benchmark datasets Ogbn-Arxiv and Ogbn-Products, we adopt the standard public splits.

Table 4.1: Micro-F1 comparison across five datasets.

LMs	GNNs	ACM		Wiki		Amazon		Ogbn-Arxiv		Ogbn-Products	
		Val.	Test	Val.	Test	Val.	Test	Val.	Test	Val.	Test
Fine-tuned LMs + GNNs											
Bert	-	77.8	75.0	69.8	69.6	90.8	90.5	71.8	69.8	82.2	69.4
	GCN	80.6	80.3	71.0	70.3	93.6	93.3	71.7	70.5	92.1	78.8
	GAT	81.0	79.9	73.5	72.8	93.7	93.6	72.9	70.6	93.0	81.5
	GraphSAGE	80.8	79.7	73.2	72.0	93.8	92.9	70.9	68.5	91.9	78.6
Roberta	-	76.2	76.4	72.1	70.6	88.9	88.5	69.8	67.1	80.8	66.5
	GCN	80.5	79.9	70.8	69.0	93.6	93.5	71.5	69.2	92.1	78.0
	GAT	80.9	80.1	73.7	73.0	94.0	93.5	72.6	70.3	92.8	81.3
	GraphSAGE	80.9	79.5	73.2	70.4	94.3	94.1	70.2	67.6	92.4	78.6
Specialized frameworks for text-attributed graphs											
MPAD		80.1	78.9	68.8	68.0	93.1	92.8	68.2	63.9	77.6	63.9
GLEM		81.4	79.8	73.6	71.2	94.5	94.3	76.9	75.9	93.2	84.7
GraphFormers		75.3	75.1	66.8	67.5	85.6	86.4	70.4	68.4	75.0	60.3
LLAGA		77.2	77.5	71.7	72.0	90.1	90.8	77.5	76.1	92.4	84.2
InstructGLM		75.4	74.5	72.2	70.6	94.3	94.2	76.9	76.2	-	-
GDL4LLM		**81.9**	**81.4**	**74.3**	**73.2**	**94.6**	**94.6**	**77.7**	**76.5**	**93.9**	**85.4**
Fine-tuned LLMs + GNNs											
GraphAdapter	-	80.8	80.4	71.9	71.7	94.1	93.4	77.3	76.5	-	-
Llama3-8b	-	80.7	80.6	71.9	71.2	92.0	91.6	73.0	72.5	-	-
Llama3-8b	GraphSAGE	82.0	81.3	72.8	73.0	93.1	92.8	76.8	76.1	-	-
GDL4LLM	**w/ attr**	**83.9**	**82.8**	**74.0**	**73.4**	**95.8**	**95.5**	**77.6**	**76.6**	-	-

4.4.2 Baselines and Variants

We benchmark against three families of strong methods: (i) Classical GNN-based models: GCN [31], GraphSAGE [16], and GAT [66]. (ii) Frameworks tailored for text-attributed graphs: MPAD [55], GLEM [106], GraphFormers [86], LLAGA [7], InstructGLM [88], and GraphAdapter [25]. (iii) Hybrid LM-GNN pipelines combining fine-tuned LMs with graph encoders.

We also include ablations of our approach: (i) *GDL4LLM w/o pre-train*: removes the graph language pre-training while keeping structure-aware fine-tuning; (ii) *GDL4LLM w/ attr*: augments prompts with node textual attributes; (iii) *GDL4LLM w/ attr w/o*

Table 4.2: Dataset statistics.

Dataset	#Nodes	#Edges	#Classes
ACM	48,579	193,034	9
Wiki	36,501	1,190,369	10
Amazon	50,000	632,802	7
Ogbn-Arxiv	169,343	1,166,243	40
Ogbn-Products	2,449,029	61,859,140	47

Table 4.3: Macro-F1 performance comparison across five datasets.

LMs	**GNNs**	**ACM**		**Wiki**		**Amazon**		**Ogbn-Arxiv**		**Ogbn-Products**	
		Val.	Test	Val.	Test	Val.	Test	Val.	Test	Val.	Test
Fine-tuned LMs + GNNs											
Bert	-	73.6	69.0	59.7	58.4	87.7	86.9	52.5	50.8	59.2	46.2
	GCN	75.9	74.6	61.0	58.7	91.5	90.1	53.3	51.6	69.5	55.8
	GAT	76.2	74.1	62.3	60.8	91.6	90.6	55.6	53.7	70.4	56.8
	GraphSAGE	76.3	74.1	61.8	59.5	91.8	90.0	50.2	48.4	65.7	54.0
Roberta	-	71.2	70.7	60.4	57.6	85.8	83.9	52.4	50.6	58.0	44.8
	GCN	75.6	74.1	60.0	56.1	91.3	90.7	53.1	51.8	68.4	55.0
	GAT	76.1	74.2	62.6	61.2	91.6	90.5	55.7	53.4	69.2	55.5
	GraphSAGE	76.0	74.1	61.2	57.5	90.6	90.4	49.1	48.4	64.8	53.7
Specialized Frameworks for Text-Attributed Graphs											
MPAD		72.2	71.6	56.2	53.9	90.4	88.6	48.6	43.5	57.6	43.2
GLEM		76.6	73.9	62.8	58.3	91.8	90.9	57.4	55.9	73.6	64.7
GraphFormers		71.0	65.4	53.5	51.2	85.2	82.2	49.4	44.8	54.3	42.1
LLAGA		72.8	72.1	60.9	60.5	89.5	88.6	60.4	60.2	72.1	64.3
InstructGLM		70.1	68.9	62.1	58.1	90.8	89.6	60.1	59.8	-	-
GDL4LLM		**77.2**	**75.6**	**63.0**	**61.5**	**92.1**	**91.3**	**60.9**	**60.5**	**74.3**	**65.5**
Fine-tuned LLMs + GNNs											
GraphAdapter	-	77.4	75.7	61.6	59.9	91.7	90.0	**60.9**	60.4	-	-
Llama3-8b	-	75.9	73.8	61.7	59.0	88.9	88.3	56.5	54.7	-	-
Llama3-8b	GraphSAGE	78.5	76.4	62.0	60.6	90.9	89.6	59.6	60.0	-	-
GDL4LLM	**w/ attr**	**79.7**	**77.7**	**63.0**	**61.5**	**93.1**	**92.4**	60.6	**60.5**	**-**	**-**

pre-train: combines attribute augmentation with no pre-training.

4.4.3 Implementation Details

We conduct all experiments on a server equipped with eight NVIDIA A100 GPUs, each with 80 GB of memory. To ensure a fair comparison, we obtain baseline results using their official publicly released implementations. Our framework is implemented in `PyTorch` with the `transformers` library.

Hyperparameters are selected via grid search. In particular, the graph sentence length l and the number of graph sentences k are tuned over the sets $\{2, 3, 4, 5\}$ and $\{2, 4, 6, 8, 10\}$, respectively, and the best configuration is chosen according to performance on the validation set. For LoRA, we fix the dropout rate, rank, and scaling factor α to 0.2, 8, and 16, respectively. We apply a weight decay of 1e$-$2 to mitigate overfitting. The proposed framework is trained with a learning rate of 1e$-$4 and a batch size of 32, using the Adam optimizer and early stopping based on micro accuracy on the validation set. And we also report the macro accuracy on the test set. In the main comparison, we adopt Llama2-7B [65] as the backbone LLM, and additionally employ Llama3-8B to investigate the effect of varying the backbone.

4.4.4 Main Results (RQ1)

Table 4.1 lists Micro-F1, and Table 4.3 lists Macro-F1 results. They demonstrate the same trend. We compare against three broad groups and summarize takeaways:

LM-GNN hybrids. These pipelines benefit from message passing in GNNs and from LM-derived features, but their overall performance hinges on external aggregation rather than the LM's internal capacity to reason over higher-order neighborhoods. The limited structural signal present in raw attributes further constrains them.

Description-based methods. Encoding neighborhoods as natural language descriptions tends to inflate prompt length, often restricting models to first-order or second-order nodes in practice. By contrast, GDL4LLM compresses structure into

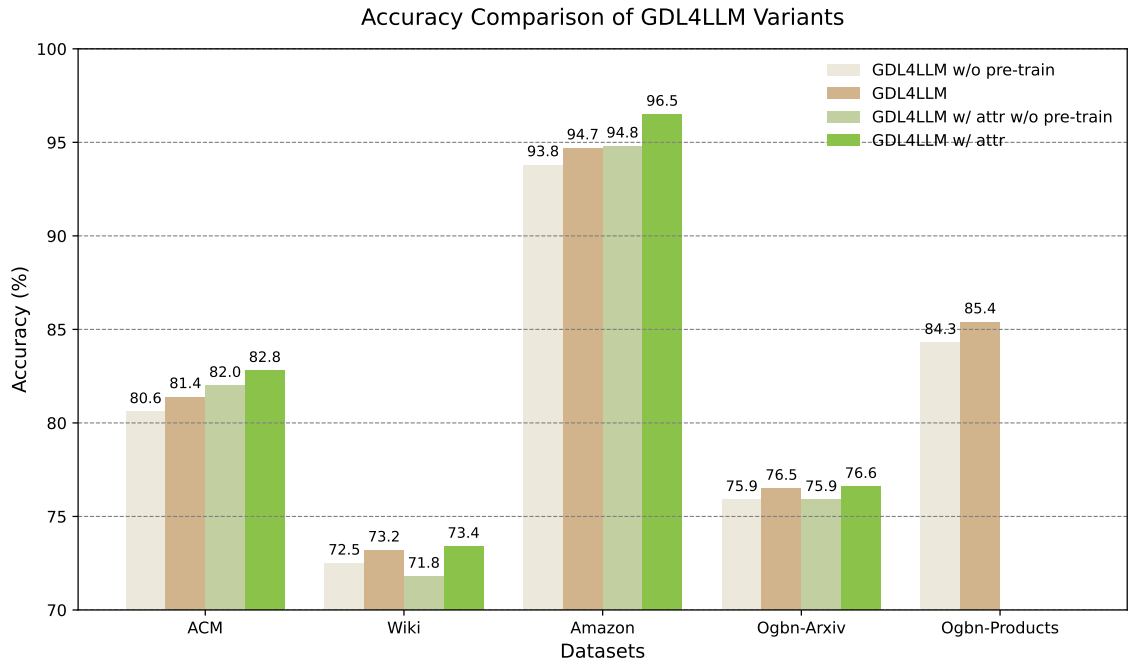


Figure 4.2: Test accuracy of GDL4LLM variants across five datasets.

compact graph tokens and scales to higher-order nodes, leading to consistent improvements.

LLM-enhanced GNNs. Although LLMs boost attribute embeddings, the integration between LLM reasoning and graph aggregation is typically shallow, which limits modeling of multi-hop dependencies. Treating structure as a language, GDL4LLM marries structural modeling and semantic understanding within a single LLM, yielding stronger results.

4.4.5 Ablation Study (RQ2)

We analyze the effects of pre-training and attribute-aware prompting. Figure 4.2 report the results.

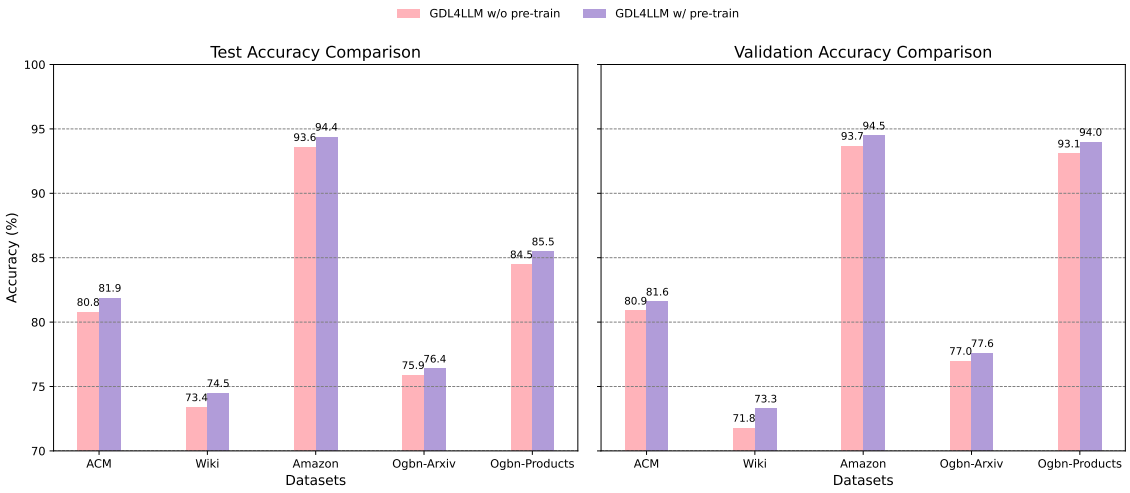


Figure 4.3: Validation and test accuracy of GDL4LLM with Llama3-8B across five datasets.

Pre-training on graph sentences substantially improves downstream accuracy, especially when combined with attributes (*GDL4LLM w/ attr*). This mirrors classic LM pre-training benefits: the model develops an internalized understanding of token transitions reflective of graph connectivity.

Incorporating textual attributes into the prompts further improves performance by adding semantic information that is not captured by the graph structure alone. Taken together, the structural knowledge learned during pre-training and the extra semantics from attributes work in a complementary way: pre-training brings in an inductive bias that is aware of node connectivity, while the attributes strengthen class separation by making use of the node attributes.

4.4.6 Backbones and Hyperparameters (RQ3)

Backbones. We replace Llama-2 with Llama-3 backbones in Figure 4.3. Llama-3 consistently yields higher scores, likely due to architectural and pre-training data

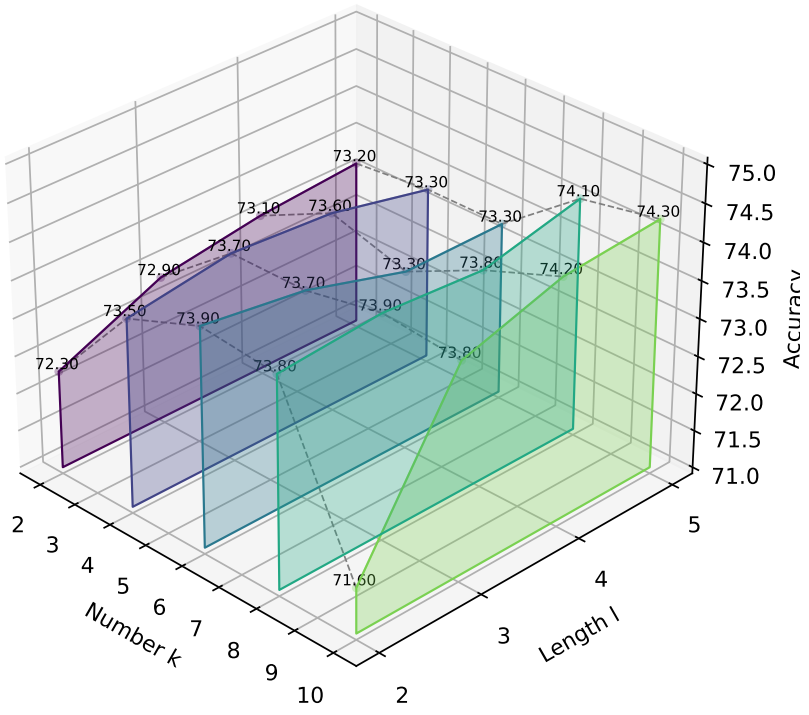


Figure 4.4: Effect of sentence length l and number k on Wiki dataset.

improvements over Llama-2. Ablations on Llama-3 reaffirm that our graph language pre-training is beneficial across different LLM backbones.

Hyperparameters. We study the sentence length l and the number of sampled sentences k. As shown in Figure 4.4, performance saturates around l=5 and k=10. This indicates that compact corpora suffice to capture multi-hop neighbors: with l=5, fourth-order context is modeled, whereas typical GNN depths, e.g., two layers emphasize second-order neighbors [10, 59].

4.4.7 Efficiency Analysis (RQ4)

We compare token usage and runtime with description-centric approaches. Table 4.4 shows that GDL4LLM dramatically reduces tokens while modeling the same or higher structural order, outperforming InstructGLM and LLAGA-HO. LLAGA-HO compresses neighborhoods via GNN summarization but still relies on verbose natural

Table 4.4: Token budget vs. modeled structural order (Token/(order)).

LLMs -	ACM		Wiki		Amazon		Ogbn-Arxiv		Ogbn-Products	
	Val.	**Test**	**Val.**	**Test**	**Val.**	**Test**	**Val.**	**Test**	**Val.**	**Test**
InstructGLM	146(1)	149(1)	1024(2)	319(1)	532(2)	538(2)	460(0)	463(0)	-	-
LLAGA-HO	155(4)	155(4)	130(4)	130(4)	140(4)	140(4)	492(4)	492(4)	370(4)	370(4)
GDL4LLM	54(4)	54(4)	72(4)	72(4)	72(4)	72(4)	106(4)	106(4)	106(4)	106(4)
Reduction (%)	**63.01**	**63.01**	**44.62**	**44.62**	**48.57**	**48.57**	**76.96**	**76.96**	**71.35**	**71.35**

Table 4.5: Training and inference time (hh:mm:ss).

LLMs	**Stage**	**ACM**	**Wiki**	**Amazon**	**Ogbn-Arxiv**	**Ogbn-Products**
InstructGLM	**Train**	5:59:04	4:30:16	6:07:02	102:29:23	N/A
	Test	0:06:41	0:05:01	0:07:08	0:42:20	N/A
LLAGA-HO	**Train**	**0:47:26**	0:35:18	0:49:29	3:09:03	5:33:11
	Test	0:04:41	0:01:57	0:04:04	0:44:12	143:01:12
GDL4LLM	**Prep**	0:00:03	0:00:58	0:00:05	0:00:28	0:35:02
	Train	1:05:23	**0:34:03**	**0:32:15**	**2:40:06**	**5:03:39**
	Test	**0:02:06**	**0:01:17**	**0:01:45**	**0:06:44**	**6:29:13**

language descriptions; by replacing descriptions with graph tokens, GDL4LLM remains succinct and scalable to higher-order context.

The efficiency benefits of GDL4LLM are clearly illustrated in Table 4.5, where it achieves faster training and inference on most datasets. On the ACM dataset, GDL4LLM finishes inference in 2:06, while InstructGLM and LLAGA-HO require 6:41 and 4:41, respectively. On the larger Ogbn-Arxiv dataset, GDL4LLM completes inference in 6:44, compared with 42:00 for InstructGLM and 44:12 for LLAGA-HO. These results indicate that GDL4LLM can greatly cut both token consumption and runtime, while still modeling high-order structure effectively.

Furthermore, the data preprocessing time required by GDL4LLM is marginal across all datasets. It is worth noting that this preprocessing is a one-time offline procedure. Therefore, it does not significantly increase the overall time cost, further confirming the practical efficiency of our proposed approach compared to the baselines.

4.5 Summary

In this work, we point out two key drawbacks of relying only on natural language to model text-attributed graphs with LLMs: (i) graph descriptions quickly become long and redundant, especially when high-order neighbors are involved; and (ii) text attributes alone do not contain enough structural signals, which makes it hard for LLMs to learn high-order graph structure. We further focus on two core challenges: (i) LLMs lack an internal mechanism similar to message passing in GNNs, and (ii) there exist strong dependencies between high-order neighbors and the target node across different orders. To tackle these issues, we introduce the GDL4LLM framework, which lets LLMs handle text-attributed graphs as if they are learning a new language. GDL4LLM has two stages: (i) we build a graph language corpus and pre-train LLMs on it so they can better understand graph structures, and (ii) we then fine-tune LLMs with this corpus to generate compact descriptions of subgraphs centered on target nodes. In this way, LLMs can gradually learn and use graph structure at different orders. Extensive experiments show that GDL4LLM is both efficient and effective for modeling text-attributed graphs in various downstream tasks.

Limitations. However, while GDL4LLM achieves superior performance by unifying semantics and structure, it introduces a critical challenge regarding deployment efficiency. As analyzed in our experiments, direct LLM inference on graphs faces two major bottlenecks: combinatorial explosion of paths in dense subgraphs and unacceptable inference latency due to auto-regressive generation. These computational constraints make real-time deployment difficult. This limitation motivates the work in the next chapter, where we will explore how to distill the structural knowledge learned by LLMs into efficient embedding-based inference models, aiming to balance the reasoning power of LLMs with the speed requirements of industrial systems.

Chapter 5

LLM Collaborative Filtering: User-Item Graph as New Language

5.1 Introduction

The Chapter 1 introduces multi-order collaborative information is central to CF and classical graph-based CF models such as LightGCN [18] capture collaborative information via iterative neighborhood aggregation on the user–item interaction graph. And as discussed in Section 2.4, to adapt LLMs to CF, recent adaptations translate a user's immediate interactions into natural language graph descriptions, falling into two broad families: **description reasoning-based** and **embedding injection-based** methods. And these approaches suffer from one or both of the following issues: **loss of multi-order collaboration** and **slow inference**.

While recent methods, e.g., LLMEmb [45], LLM-CF [61], improve inference by building user/item profiles [23] with LLMs, they do not model the user–item graph with LLMs and do not completely solve these limitations. Given the efficacy of graphs for capturing multi-order collaboration [73], leveraging LLMs' learning ability over graph structure is compelling but hindered by:

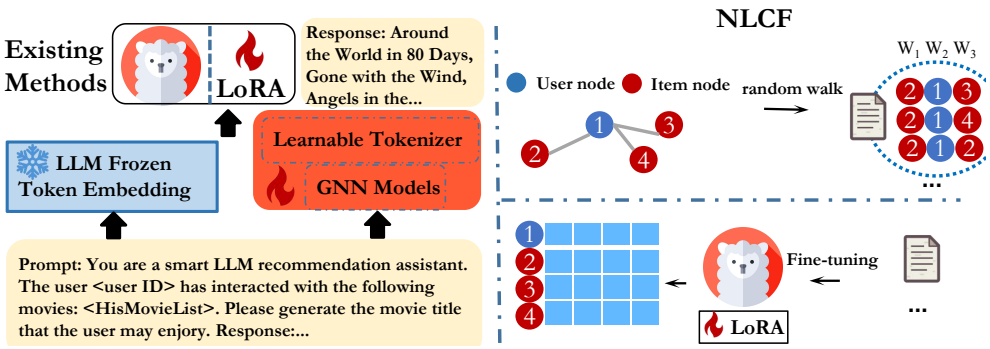


Figure 5.1: Pipeline comparison between prior methods and NLCF. NLCF attains efficiency by training on short graph sentences and serving via precomputed user-item embeddings.

(1) **Structural mismatch:** User–item graphs are irregular 2D structures, whereas LLMs process 1D sequences. Unlike graph-based CF models that aggregate over neighborhoods [80], LLMs lack native graph inductive bias.

(2) **Compute constraints:** The count of multi-order neighbors grows geometrically, yet LLM fine-tuning and serving are costly at scale. Meeting real-world latency targets further constrains deployment.

We introduce New Language Collaborative Filtering (NLCF), a framework that treats the user-item graph as a new language to let LLMs efficiently learn collaborative embeddings. As shown in Figure 5.1, NLCF transforms the graph into a compact corpus whose empirical token transition probabilities converge to graph transition probabilities. This design supports multi-order interaction modeling. NLCF then balances frequent and rare multi-order interactions via similarity-based sampling. The sampled corpus is used to fine-tune an LLM, whose hidden states are aggregated to construct user/item collaborative embeddings. Training is efficient due to concise graph sentences, and inference is efficient via embedding retrieval rather than token generation. To clearly distinguish our proposed framework from these existing generation-based LLM CF paradigms, we provide a conceptual comparison in Table 5.1. And our

contributions are as follows:

- We propose a framework NLCF, which integrates LLMs with CF by converting the user–item graph as a new language, enabling LLMs to learn collaborative user/item embeddings.
- We develop two core modules: (i) a graph corpus collection module that converts the graph into compact sentences and applies similarity-based sampling; and (ii) a collaborative embedding construction module that fine-tunes LLMs and aggregates hidden states for efficient retrieval.
- Extensive experiments on three datasets show consistent gains over graph-based CF and LLM-based baselines, with notable efficiency improvements. Online A/B tests further validate industrial effectiveness.

Table 5.1: Conceptual comparison between generation-based LLM CF and the proposed NLCF framework.

Feature	Generation-based LLM CF	NLCF (Ours)
Inference Mechanism	Autoregressive text generation	Embedding-based dot-product search
Inference Latency	High (Depends on output length)	Low (Efficient similarity search)
Inference Versatility	High (Free text)	Focused (Ranked item lists)
Input Representation	Natural language prompts	Compact graph sentences (Random walks)
Structure Modeling	Implicit reasoning via descriptions	Explicit multi-order interaction encoding
Hallucination Issue	Prone to generating non-existent items	Free (Constrained to candidate set)
Deployment	Hard (Slow for real-time serving)	Easy (Compatible with ANN indices)

5.2 Preliminary

Notation. Let the user–item graph be $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \mathcal{U} \cup \mathcal{I}$ is the union of users $\mathcal{U}$ and items $\mathcal{I}$, indexed by $\{v_1, \ldots, v_{|\mathcal{V}|}\}$. Edges $\mathcal{E} \subseteq \mathcal{U} \times \mathcal{I}$ encode interactions,

with adjacency $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$:

$$\mathbf{A}_{ui} = \begin{cases} 1, & \text{if } (v_u, v_i) \in \mathcal{E}, \\ 0, & \text{otherwise.} \end{cases} \tag{5.1}$$

The bipartite nature implies zero entries for user–user and item–item pairs. We fine-tune an LLM on $\mathcal{G}$ with LoRA [21] parameters $\hat{\mathbf{W}}$ to learn user embeddings $\mathbf{h}_{v_u}$ and item embeddings $\mathbf{h}_{v_i}$.

Low-Rank Adaptation. LoRA fine-tunes LLMs efficiently by introducing trainable low-rank matrices [21]. Given a weight matrix $\mathbf{W} \in \mathbb{R}^{d\times d'}$, LoRA augments it as $\hat{\mathbf{W}} = \mathbf{W} + BA$ during adaptation.

Collaborative Filtering. CF learns user/item embeddings from user–item interactions; multi-order collaborative signals in $\mathcal{G}$ [18] are crucial. The forward process updates embeddings $\mathbf{h}_{v_u}$, $\mathbf{h}_{v_i}$ via

$$\mathbf{H}^{(k)} = f(\mathbf{A}, \mathbf{H}), \tag{5.2}$$

where $f(\cdot)$ is a GNN capturing higher-order collaboration. To do recommendations, preference scores over $\mathcal{I}$ use

$$\hat{y}_{ui} = \mathbf{h}_{v_u}^{(k)\top} \mathbf{h}_{v_i}^{(k)}. \tag{5.3}$$

And we retrieve Top-N items per user.

5.3 New Language Collaborative Filtering

We present NLCF, which lets LLMs learn collaborative embeddings by framing the user–item graph as a language. As in Figure 5.2, NLCF has two modules: (i) **Graph Corpus Collection**, which converts the graph into a sentence corpus encoding multi-order interactions and reduces it via similarity-based sampling; and (ii) **Collaborative Embedding Construction**, which fine-tunes the LLM on the corpus and aggregates hidden states to form user/item embeddings.

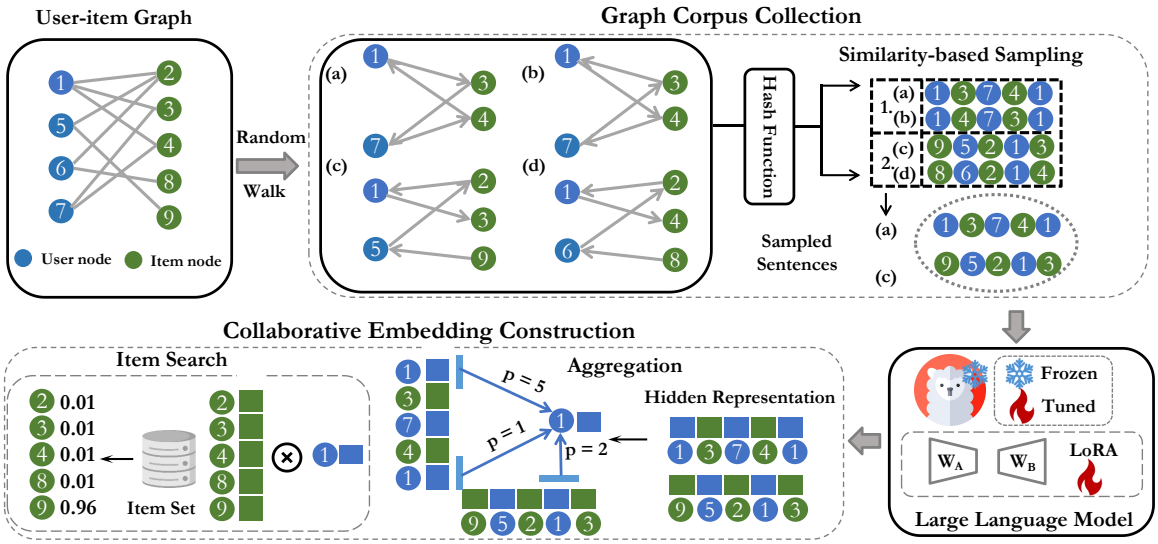


Figure 5.2: Overall NLCF pipeline. Top: collect corpus via random walks followed by similarity-based sampling. Bottom: fine-tune LLM on the corpus, then aggregate hidden states to build collaborative user/item embeddings for efficient retrieval.

5.3.1 Graph Corpus Collection

We first map graph elements to language elements, then detail a similarity-based sampling method.

Definition of Graph Language-related Concepts.

Graph tokens from users/items. We treat nodes $\mathcal{V}$ as unique tokens in the new vocabulary, extending the tokenizer with randomly initialized embeddings.

Graph sentences from paths. Any connected node sequence, a path, is a sentence. For example, in Figure 5.2, $\mathbf{s}_1 = \{v_9, v_5, v_2, v_1, v_3\}$ is a sentence. Such paths encode multi-order collaboration: e.g., v_1 may interact with v_9 because second-order neighbor v_5 interacted with v_9. The initial corpus $\mathcal{S} = \{\mathbf{s}_1, \mathbf{s}_2, \dots\}$ consists of these sentences.

We extract sentences with random walks [14] using transition probabilities:

$$P_g(v_i \mid v_u) = \begin{cases} \frac{\mathbf{A}_{ui}}{\sum_{j \in \mathcal{N}(v_u)} \mathbf{A}_{uj}}, & v_i \in \mathcal{N}(v_u), \\ 0, & (v_u, v_i) \notin \mathcal{E}, \end{cases} \tag{5.4}$$

where $\mathcal{N}(v_u)$ is the neighbor set of v_u. Each walk continues until reaching length l. This procedure ensures empirical conditional probabilities in the corpus converge to graph transition probabilities:

Theorem 5. *Let $P_s(v_i \mid v_u)$ be the empirical conditional probability from a corpus generated by sufficiently many random walks. As the number of walks tends to infinity, $P_s(v_i \mid v_u)$ converges in probability to $P_g(v_i \mid v_u)$.*

Proof. We prove that the empirical conditional probability $P_{\mathrm{s}}(v_i \mid v_u)$ computed from the corpus converges to the graph transition probability $P_{\mathrm{g}}(v_i \mid v_u)$.

Let $N_{\mathrm{s}}(v_u \to v_i)$ be the number of observed transitions from v_u to v_i in the initial corpus. The empirical probability is

$$P_{\mathrm{s}}(v_i \mid v_u) = \frac{N_{\mathrm{s}}(v_u \to v_i)}{\sum_{j \in \hat{\mathcal{N}}(v_u)} N_{\mathrm{s}}(v_u \to v_j)},$$

where $\hat{\mathcal{N}}(v_u)$ denotes multi-order neighbors within walk length l. The expected transition count is

$$\mathbb{E}[N_{\mathrm{s}}(v_u \to v_i)] = d_u \cdot l \cdot P_{\mathrm{g}}(v_i \mid v_u).$$

By the Law of Large Numbers, as $d_u \to \infty$ or we sample $k \cdot d_u$ walks with $k \to \infty$,

$$\frac{N_{\mathrm{s}}(v_u \to v_i)}{d_u \cdot l} \to P_{\mathrm{g}}(v_i \mid v_u).$$

The total outgoing transitions from v_u in sampled walks satisfy

$$\mathbb{E}\left[\sum_{j \in \hat{\mathcal{N}}(v_u)} N_{\mathrm{s}}(v_u \to v_j)\right] = \sum_{j \in \hat{\mathcal{N}}(v_u)} \mathbb{E}[N_{\mathrm{s}}(v_u \to v_j)] = d_u \cdot l.$$

Substituting into $P_{\mathrm{s}}(v_i \mid v_u)$,

$$P_{\mathrm{s}}(v_i \mid v_u) = \frac{N_{\mathrm{s}}(v_u \to v_i)}{\sum_{j \in \hat{\mathcal{N}}(v_u)} N_{\mathrm{s}}(v_u \to v_j)} \to \frac{d_u \cdot l \cdot P_{\mathrm{g}}(v_i \mid v_u)}{d_u \cdot l} = P_{\mathrm{g}}(v_i \mid v_u).$$

Therefore, random-walk sampling aligns corpus conditional probabilities with graph transition probabilities. □

To sufficiently capture collaboration, we initiate $d_u = \sum \mathbf{A}_{u:}$ walks from each node v_u, yielding the initial corpus $\mathcal{S}$.

Similarity-based Sampling for Graph Corpus Reduction.

Corpus size scales with both node count and average degree. Naive downsampling, e.g., random or degree-based sampling, ignores local density. Dense regions produce many overlapping walks. Sampling strategies for LLM recommendation [41, 43] often target a small candidate set and are incompatible with whole-item set retrieval [113]. Other approaches rely on extra LLM processing [111] or data integration [75], which is costly.

We propose similarity-based sampling: cluster similar sentences by overlap, then sample more interactions in denser clusters to reduce redundancy while preserving essentials. We measure overlap via Jaccard similarity:

$$S_{\text{similarity}}(\mathbf{s}_p, \mathbf{s}_q) = \frac{|\mathbf{s}_p \cap \mathbf{s}_q|}{|\mathbf{s}_p \cup \mathbf{s}_q|}. \tag{5.5}$$

Sentences form clusters if pairwise similarity exceeds a threshold t:

$$\mathcal{C}_i = \{\mathbf{s}_p \in \mathcal{S} : S_{\text{similarity}}(\mathbf{s}_p, \mathbf{s}_q) \geq t,\ \forall \mathbf{s}_q \in \mathcal{C}_i\}. \tag{5.6}$$

Cluster size $|\mathcal{C}_i|$ reflects density. We assign each sentence $\mathbf{s}_q \in \mathcal{C}_i$ a sampling probability proportional to cluster size and normalized across clusters:

$$P(\mathbf{s}_q) = \frac{|\mathcal{C}_i|}{\sum_{j=1}^{m} |\mathcal{C}_j|}, \quad \mathbf{s}_q \in \mathcal{C}_i, \tag{5.7}$$

where m is the number of clusters. Given a budget ratio $\alpha \in (0, 1]$, we sample:

$$\mathcal{S}' = \{\mathbf{s}_q \sim P(\mathbf{s}_q) : \mathbf{s}_q \in \mathcal{S},\ |\mathcal{S}'| = \alpha|\mathcal{S}|\}. \tag{5.8}$$

This stratified sampling strategy balances the representation of different clusters while keeping the number of sampled sentences within a fixed budget, thereby controlling both redundancy and computational cost.

Directly computing pairwise Jaccard similarities via Eq. (5.5) would yield fine-grained clusters, but requires $O(|S|^2)$ comparisons and quickly becomes infeasible for large corpora. To avoid this quadratic cost, we instead use MinHash to approximate Jaccard similarity between sentences. MinHash provides efficient, unbiased similarity estimates from compact signatures, enabling scalable clustering. These properties are proved in the following theorem:

Theorem 6. *For two sentences s_p, s_q with Jaccard $J(s_p, s_q)$, the MinHash estimator $\hat{J}(s_p, s_q)$ based on k independent hash functions is unbiased with variance $\frac{J(s_p,s_q)(1-J(s_p,s_q))}{k}$.*

Proof. This proof first introduces the construction of MinHash signatures for graph sentences and establishes their relationship with Jaccard similarity under both single and multiple hash function scenarios. We then demonstrate how MinHash signatures achieve more efficient similarity estimation compared to pairwise comparison.

Let $\mathcal{H} = \{h_1, ..., h_k\}$ be a set of independent hash functions. The hash signature for a graph sentence s_p is defined as:

$$\text{sig}(s_p) = [\min_{v_i \in s_p} h_1(v_i), \min_{v_i \in s_p} h_2(v_i), ..., \min_{v_i \in s_p} h_k(v_i)].$$

For $k = 1$, we establish the following lemma: Given a single hash function h and graph sentences s_p, s_q, the probability of their MinHash signatures being equal is equivalent to their Jaccard similarity:

$$P(\text{sig}(s_p) = \text{sig}(s_q)) = J(s_p, s_q) = \frac{|s_p \cap s_q|}{|s_p \cup s_q|}.$$

Consider a random hash function h that induces a random permutation by sorting elements from $s_p \cup s_q$ by their hash values. Let $\text{sig}(s_p) = \min_{v_i \in s_p} h(v_i)$ represent the

first element of s_p in this permutation. The equality $\text{sig}(s_p) = \text{sig}(s_q)$ occurs if and only if the first element belongs to $s_p \cap s_q$. Due to the random permutation property, this probability equals the fraction of shared elements: $P(\text{sig}(s_p) = \text{sig}(s_q)) = \frac{|s_p \cap s_q|}{|s_p \cup s_q|} = J(s_p, s_q)$.

For multiple hash functions, we construct an estimator using MinHash signatures to approximate Jaccard similarity. Let $I(\min_{v_i \in s_p} h_j(v_i) = \min_{v_i \in s_q} h_j(v_i))$ denote an Bernoulli indicator. The estimator is defined as:

$$\hat{J}(s_p, s_q) = \frac{1}{k} \sum_{j=1}^{k} I(\min_{v_i \in s_p} h_j(v_i) = \min_{v_i \in s_q} h_j(v_i)).$$

Given k independent hash functions, the MinHash estimator $\hat{J}(s_p, s_q)$ is unbiased with variance $\frac{J(s_p,s_q)(1-J(s_p,s_q))}{k}$.

Since each indicator is an independent Bernoulli random variable, by linearity of expectation:

$$E[\hat{J}(s_p, s_q)] = E[\frac{1}{k} \sum_{j=1}^{k} I(\min_{v_i \in s_p} h_j(v_i) = \min_{v_i \in s_q} h_j(v_i))] = J(s_p, s_q)$$

$$Var[\hat{J}(s_p, s_q)] = \frac{J(s_p, s_q)(1 - J(s_p, s_q))}{k}.$$

Thus, $\hat{J}(s_p, s_q)$ provides an unbiased estimation of Jaccard similarity. While traditional Jaccard similarity computation requires $\mathcal{O}(|\mathcal{S}|^2)$ operations, MinHash acceleration reduces this to $\mathcal{O}(|\mathcal{S}|)$ by storing signatures in a hash table and retrieving graph sentences whose collision frequencies exceed a specified threshold.

□

Through the above theorem, we can conclude that using compact signatures and hash tables yields expected similarity search complexity $\mathcal{O}(|\mathcal{S}|)$, making large-scale corpus curation more practical.

Theoretical Analyses about Connection between Graph Corpus and Collaborative Information.

We further quantify how the one-dimensional corpus preserves multi-order collaborative information from the two-dimensional graph:

Theorem 7. *Let w denote the importance of node v_i to node v_u, as measured by the gradient norm of a GCN, and let $w' = \frac{n'}{\hat{n}}$ represent the empirical co-occurrence ratio, where n' is the number of graph sentences containing both nodes v_u and v_i, and $\hat{n}$ is the number of graph sentences containing node v_u. The standard error of $|w - w'|$, is bounded by $O\left(\frac{1}{\hat{n}}\right)$.*

Proof. Let $h_{v_u}^{(l)}$ denote the hidden feature learned by GCN [16], defined as:

$$h_{v_u}^{(l)} = \text{ReLU}\left(\frac{1}{d_u} \cdot \sum_{v_j \in \mathcal{N}(v_u)} \mathbf{W}_l h_{v_j}^{(l-1)}\right)$$

For simplicity, we remove the non-linear activation function and GCN weights from the GCN aggregation in this proof. However, the theorem still holds when these components are included. The gradient of $h_{v_u}^{(l)}$ with respect to $h_{v_i}^{(0)}$ is given by:

$$\frac{\partial h_{v_u}^{(l)}}{\partial h_{v_i}^{(0)}} = \frac{1}{d_u} \cdot \sum_{v_j \in \mathcal{N}(v_u)} \frac{\partial h_{v_j}^{(l-1)}}{\partial h_{v_i}^{(0)}}.$$

We iteratively expand this formula using the chain rule to get:

$$\frac{\partial h_{v_u}^{(l)}}{\partial h_{v_i}^{(0)}} = \sum_{p=1}^{n} \left[\frac{\partial h_{v_u}^{(l)}}{\partial h_{v_i}^{(0)}}\right]_p = \sum_{p=1}^{n} \prod_{q=1}^{l'} \frac{1}{d_{v_{q|p}}} = w,$$

where n is the number of paths containing both nodes v_i and v_u, l' is the length of the current path p, $d_{v_{q|p}}$ represents the degree of the q-th node in the current path p. Since we use Eq. (5.4) and perform random walk, the probability of node v_u visiting v_i is exactly the same as the sum of probabilities for all paths shown above.

However, it is computationally impractical to retrieve all such paths for every pair of nodes, especially when similarity-based sampling is used. Let X_p be an indicator Bernoulli variable for the p-th walk in practice, which shows whether the walk successfully reaches node v_i starting from v_u:

$$X_p = \begin{cases} 1 & \text{if the walk reaches } v_i, \\ 0 & \text{otherwise.} \end{cases}$$

From the previous theorem, the following expectation and variance properties hold: $E[X_p] = w$, and $Var(X_p) = w(1-w)$. Now, if there are $\hat{n}$ paths from node v_u to node v_i, the empirical probability satisfies the following properties: $E\left[\frac{\sum_{p=1}^{\hat{n}} X_p}{\hat{n}}\right] = w$, and $Var\left[\frac{\sum_{p=1}^{\hat{n}} X_p}{\hat{n}}\right] = \frac{w(1-w)}{\hat{n}}$. This variance is bounded by 1. Therefore, the sampling standard error of the empirical probability w', i.e., $\frac{n'}{\hat{n}}$ encoded in the sampled corpus, satisfies:

$$|w - w'| = O\left(\frac{1}{\hat{n}}\right).$$

□

In the proof, w represents collaborative information captured by GCNs in prediction by modeling multi-order interactions, while w' denotes the importance derived from the sampled graph corpus as reflected by the empirical occurrence ratio. The theorem demonstrates that the sampled graph corpus encodes collaborative information in a manner consistent with GCNs, with the difference diminishing as the number of sampled graph sentences increases. This result provides a theoretical foundation for using graph sentences as a proxy for mining collaborative information underlying multi-order interactions.

5.3.2 Collaborative Embedding Construction

After converting the user–item graph into a collaboration-rich language corpus, we fine-tune the LLM on this corpus to absorb multi-order interactions. We then aggregate its hidden representations into user and item embeddings, enabling efficient full-item inference via embedding-based retrieval rather than online LLM generation.

Fine-tuning LLMs on the Corpus.

Multi-order collaboration allows distant tokens to inform next-token prediction within sentences. We fine-tune the LLM by maximizing next-token likelihood:

$$\mathcal{L}_{\text{pre}} = -\sum_{q=1}^{|\mathcal{S}'|}\sum_{j=1}^{|\mathbf{s}_q|} \log P(s_{q,j} \mid \mathbf{s}_{q,<j}, \mathbf{W}_p, \hat{\mathbf{W}}), \tag{5.9}$$

where $\mathbf{W}_p \in \mathbb{R}^{|V|\times d}$ is the learnable head layer. This objective encourages the LLM to internalize multi-order interactions. To control the memory usage induced by large new tokens, We employ a dynamic memory bank to control memory usage.

Dynamic Memory Bank

Fine-tuning LLMs on large vocabularies is memory-intensive: each user/item token requires its embedding and corresponding row in $\mathbf{W}_p \in \mathbb{R}^{|V|\times d}$. We design a dynamic memory bank with two optional strategies: (i) batch-wise loading of only the embeddings and head weights needed for current tokens; and (ii) sampled-softmax optimization where we update only weight vectors for tokens appearing in the batch. Together, these enable scalable fine-tuning on large datasets.

Graph Sentence Representation Aggregation.

After fine-tuning, the LLM has absorbed collaborative information from the graph corpus, but invoking it online for every request would be too slow for real-world

recommenders [53]. We therefore precompute user and item embeddings offline and use them for fast retrieval at inference. For each sentence, we compute hidden states:

$$\mathbf{h}_{q,j} = \mathrm{LLM}(\{s_{q,1}, \ldots, s_{q,j-1}\}), \tag{5.10}$$

and aggregate target node representations across graph sentences to encode multi-order interactions:

$$\begin{aligned} &\mathbf{h}_{v_u} = \sum_{p \in \mathcal{K}} \mathbf{h}_{q,p}, \\ &\mathcal{K} = \{p : \mathbf{s}_q \in \mathcal{S}',\ s_{q,p} = v_u,\ p = |\mathbf{s}_q| - k,\ 0 < k < |\mathbf{s}_q|\}\,. \end{aligned} \tag{5.11}$$

where $\mathcal{K}$ collects valid positions of v_u controlled by the hyperparameter k. Item embeddings are computed in the same way. Once embeddings are precomputed, NLCF serves recommendations via simple inner-product search using these embeddings, avoiding any online LLM computation and thus achieving low-latency inference.

5.4 NLCF Pseudocode

Algorithm 1 outlines the workflow:

In *corpus collection*, NLCF performs d_u random walks of length l per node, clusters sentences with MinHash based on threshold t, and samples with $P(s_q) \propto |\mathcal{C}_i|$ to produce $\mathcal{S}'$ of size $\alpha|\mathcal{S}|$.

In *embedding construction*, LoRA parameters $\hat{\mathbf{W}}$ are fine-tuned by minimizing $L_{\text{pre}} = -\sum_q \sum_j \log P(s_{q,j} \mid s_{q,<j}, \mathbf{W}_p, \hat{\mathbf{W}})$ with a dynamic memory bank. User/item embeddings $\mathbf{h}_{v_u}, \mathbf{h}_{v_i}$ are aggregated from positions where the tokens appear.

In *serving*, scores $\hat{y}_{ui} = \mathbf{h}_{v_u}^\top \mathbf{h}_{v_i}$ enable fast top-N retrieval.

Algorithm 1 New Language Collaborative Filtering

Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, LLM params $\mathbf{W}_p$, sampling ratio α, walk length l, threshold t, agg. offset k
Output: User/item embeddings $\mathbf{h}_v$
procedure CollectGraphCorpus($\mathcal{G}, l, \alpha, t$)
    $\mathcal{S} \leftarrow \{\text{RandomWalk}(\mathcal{G}, v, l) \mid v \in \mathcal{V},\ \text{repeated degree}(v)\}$
    Cluster $\mathcal{S}$ into $\{\mathcal{C}_i\}$ by MinHash (sim. threshold t)
    For $s_q \in \mathcal{C}_i$: $P(s_q) \leftarrow \dfrac{|\mathcal{C}_i|}{\sum_j |\mathcal{C}_j|}$
    Sample $\mathcal{S}'$ of size $\alpha|\mathcal{S}|$ from $\mathcal{S}$ w.r.t. P
    return $\mathcal{S}'$
end procedure
procedure ConstructEmbeddings($\mathcal{S}', \mathbf{W}_p$)
    Initialize LoRA params $\hat{\mathbf{W}}$
    while not converged **do**
        Sample batch $\{s_q\} \subset \mathcal{S}'$
        $L_{\text{pre}} \leftarrow -\sum_q \sum_j \log P(s_{q,j} \mid s_{q,<j}; \mathbf{W}_p, \hat{\mathbf{W}})$
        Update $\hat{\mathbf{W}}$ by minimizing L_{pre}
    end while
    for each $v \in \mathcal{V}$ **do**
        $\mathcal{K}_v \leftarrow \{(q,p) : s_{q,p} = v,\ p = |s_q| - k\}$
        $\mathbf{h}_v \leftarrow \sum_{(q,p)\in\mathcal{K}_v} \text{LLM}(s_{q,1:p-1})$
    end for
    return $\{\mathbf{h}_v\}_{v\in\mathcal{V}}$
end procedure
procedure NLCF($\mathcal{G}, \mathbf{W}_p, l, \alpha, t, k$)
    $\mathcal{S}' \leftarrow$ CollectGraphCorpus($\mathcal{G}, l, \alpha, t$)
    $\{\mathbf{h}_v\} \leftarrow$ ConstructEmbeddings($\mathcal{S}', \mathbf{W}_p$)
    return $\{\mathbf{h}_v\}$
end procedure
procedure Recommend($u, \mathcal{I}, \{\mathbf{h}_v\}, N$)
    $\hat{y}_{ui} \leftarrow \mathbf{h}_u^\top \mathbf{h}_i,\ \forall i \in \mathcal{I}$
    return Top-N items by $\hat{y}_{ui}$
end procedure

5.5 Experiments

We perform comprehensive experiments on three real-world datasets to assess both the effectiveness and efficiency of the NLCF framework. Specifically, our empirical study is designed to answer the following research questions: **RQ1**: How does NLCF compare with state-of-the-art graph-based CF models and LLM-driven recommendation methods in terms of overall performance? **RQ2**: How do algorithmic design choices, e.g., different sampling mechanisms and alternative LLM backbones, impact the performance of NLCF? **RQ3**: To what extent is NLCF sensitive to crucial hyperparameters, including the sampling ratio and the length of generated sentences? **RQ4**: How well does NLCF perform when deployed real-world settings?

5.5.1 Experimental Settings

Datasets.

Before detailing the specific datasets used in this chapter, it is important to note the rationale for selecting different data sources compared to the previous chapter. While previous chapter focuses on generic node classification tasks utilizing homogeneous graphs, e.g., citation and co-purchase graphs, this chapter addresses the CF problem.

CF tasks fundamentally differ from node classification as they require modeling user-item interactions within a bipartite graph structure to predict missing links or rank items. Standard graph benchmarks like OGB-Arxiv lack this specific user-item dichotomy. Therefore, to effectively evaluate the proposed NLCF model and ensure fair comparison with standard recommendation baselines, we employ three widely recognized datasets in the recommendation community: ML-10M, Steam, and ML-1M:

- **ML-10M** [17]: a large-scale MovieLens dataset with user–movie ratings and detailed movie metadata, titles, genres, plots, etc. It contains 2,340,369 inter-

actions from 69,428 users over 5,180 items.

- **Steam** [29]: user interactions on a gaming platform, with game metadata such as titles and descriptions. It includes 918,951 interactions from 41,008 users over 2,438 items.
- **ML-1M** [17]: a medium-scale MovieLens dataset with user–movie ratings and user/item metadata, e.g., user demographics, movie genres. It contains 370,647 interactions from 4,869 users over 1,818 items.

These datasets provide textual attributes that can be verbalized as input to LLM-based baselines. Following [44], we split each dataset into training/validation/test sets with an 8:1:1 ratio, and remove users and items in the validation and test sets that do not appear in the training set. We treat ratings ≥ 3 as positive feedback, as in prior LLM recommendation work [44], and filter out users and items with fewer than 10 interactions [71].

Baselines Methods.

We compare NLCF against two categories of recommendation models:

- **Graph-based CF.** (i) **LightGCN** [18]: a linear graph propagation model that removes nonlinear activations to learn user/item embeddings. (ii) **LightGCL** [4]: introduces SVD-based graph augmentation to alleviate sparsity and popularity bias. (iii) **HMLET** [32]: combines linear and nonlinear propagation with centrality-aware adaptation. (iv) **AFDGCF** [79]: applies dynamic feature-dimension filtering to reduce over-correlation and over-smoothing.
- **LLM-based recommendation.** (i) **TransRec** [42]: a transition-style LLM framework that models multi-facet identifiers and item transitions via textual prompts. (ii) **LLM-CF** [61]: leverages instruction tuning to inject collaborative signals and world knowledge into LLMs for recommendation. (iii)

Table 5.2: Precision with $N = 5$ and 10 on Steam, ML-10M, and ML-1M.

Model	Steam		ML-10M		ML-1M	
	Precision@5	**Precision@10**	**Precision@5**	**Precision@10**	**Precision@5**	**Precision@10**
LightGCN [18]	0.1232	0.0813	0.1559	0.1195	0.2527	0.2153
LightGCL [4]	0.0654	0.0503	0.1248	0.0911	0.1873	0.1569
HMLET [32]	0.1258	0.0884	0.1461	0.1141	<u>0.2589</u>	<u>0.2238</u>
AFDGCF [79]	<u>0.1307</u>	<u>0.0909</u>	<u>0.1575</u>	<u>0.1208</u>	0.2575	0.2223
TransRec [42]	0.0858	0.0448	0.1081	0.0603	0.1766	0.1218
LLM-CF [61]	0.1257	0.0827	0.1524	0.1132	0.2550	0.2170
LETTER [70]	0.1033	0.0690	0.1213	0.0835	0.2018	0.1645
LC-Rec [107]	0.0924	0.0606	0.1106	0.0664	0.1931	0.1536
LLMEmb [45]	0.1280	0.0862	0.1572	0.1199	0.2542	0.2164
NLCF	**0.1356**	**0.0932**	**0.1730**	**0.1281**	**0.2650**	**0.2297**

LETTER [70]: learns task-specific tokenization that fuses LLM semantics with graph-based CF signals. (iv) **LC-Rec** [107]: combines prompt-template fine-tuning with learnable tokenization for LLM-based recommendation. (v) **LLMEmb** [45]: fine-tunes LLMs on item attributes to generate reusable embeddings for downstream recommendation models.

These LLM baselines cover three major paradigms: (i) *description-based reasoning* (TransRec, LLM-CF), which directly prompts LLMs with textual descriptions for recommendation; (ii) *learnable tokenization and embedding injection* (LETTER, LC-Rec), which adapt the tokenization and input embedding layer of LLMs to better encode collaborative signals; and (iii) *attribute-based embedding generation* (LLM-CF, LLMEmb), which leverages LLMs as powerful encoders to produce item attribute embeddings for external recommendation models. We also compare our similarity-based sampling strategy with a simple random sampling variant to isolate the benefit of structured sampling.

Implementation details.

All baselines are implemented using publicly available code and run on a single NVIDIA A100-SXM4-40GB GPU under Ubuntu 20.04.6, with Python 3.9.22 and Transformers 4.45.2. CUDA 12.8 is used to manage GPU kernels. For fair comparison, we measure both training time and inference time on the same hardware.

For graph-based CF methods, we follow their default training pipelines where possible, and adopt a batch size of 4,096. For NLCF, we use a batch size of 256 and a learning rate of 1×10^{-4}. For LLM-based baselines, the batch size is maximized subject to GPU memory and prompt-length constraints; models that rely on precomputed embeddings use similarity search at inference time, while generation-based methods decode item identifiers via the LLM.

We tune hyperparameters on the validation set via grid search. For NLCF, key hyperparameters include the walk length $l \in \{2, 3, 4, 5\}$, the sampling temperature $t \in \{0.25, 0.5, 0.75, 1\}$, and the aggregation position is fixed at $k = 1$ unless otherwise specified. The sampling proportion is set to $\alpha = 75\%$ in main experiments, and varied in ablation studies. Our main comparisons use `LLaMA-3.1-1B` as the LLM backbone; to evaluate generalization across model sizes and families, we further test `LLaMA-2-7B`, `LLaMA-3.2-3B`, and `LLaMA-3.1-8B`.

Each method is run five times with different random seeds, and we report the mean performance. Offline evaluation adopts full-ranking Precision@N and NDCG@N [115, 99, 110] with $N \in \{5, 10\}$. For online A/B testing, we track both effectiveness and system efficiency using the following metrics:

Precision fraction of relevant items in the top-K list.

Normalized Discounted Cumulative Gain (NDCG) ranking quality that discounts by position.

Page Click-Through Rate (PCTR) overall clicks divided by impressions.

Table 5.3: NDCG with $N = 5$ and 10 on Steam, ML-10M, and ML-1M.

Model	Steam		ML-10M		ML-1M	
	NDCG@5	**NDCG@10**	**NDCG@5**	**NDCG@10**	**NDCG@5**	**NDCG@10**
LightGCN [18]	0.2744	0.2790	0.1988	0.2007	0.2695	0.2436
LightGCL [4]	0.2623	0.2807	0.1944	<u>0.2065</u>	0.2112	0.2045
HMLET [32]	0.2730	0.2853	0.1919	0.1950	0.2723	<u>0.2503</u>
AFDGCF [79]	<u>0.2809</u>	<u>0.2897</u>	0.2002	0.2004	0.2694	0.2484
TransRec [42]	0.2796	0.2843	0.1961	0.1975	0.2706	0.2435
LLM-CF [61]	0.2772	0.2829	0.1976	0.1993	<u>0.2735</u>	0.2478
LETTER [70]	0.2717	0.2776	0.1928	0.1883	0.2683	0.2350
LC-Rec [107]	0.2615	0.2682	0.1865	0.1822	0.2626	0.2321
LLMEmb [45]	0.2785	0.2858	<u>0.2042</u>	0.2034	0.2711	0.2446
NLCF	**0.2864**	**0.2948**	**0.2171**	**0.2147**	**0.2796**	**0.2558**

User Click-through Rate (UCTR) fraction of users who clicked.

Gross Merchandise Value (GMV) total purchase value.

Response Time (ResTime) average end-to-end response time from request to returned results.

5.5.2 Main Comparison (RQ1)

Tables 5.3 and 5.2 summarize results with two observations. First, LLM-based methods do not consistently surpass strong graph-based CF across metrics. Integrating collaborative information via prompts remains challenging; approaches injecting precomputed embeddings, e.g., LC-Rec, LETTER, inherit information loss from not directly modeling multi-order interactions, which likely hurts as k increases, especially for Precision. Second, NLCF achieves the best overall performance on all datasets, indicating a paradigm shift: instead of depending on graph-based CF for multi-order signals, NLCF enables LLMs to learn collaboration directly from the graph corpus.

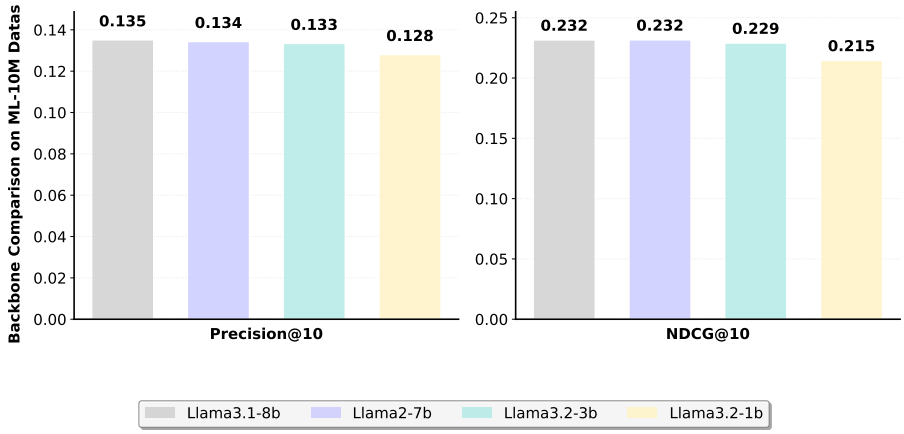


Figure 5.3: Effect of backbone parameter scale on ML-10M dataset.

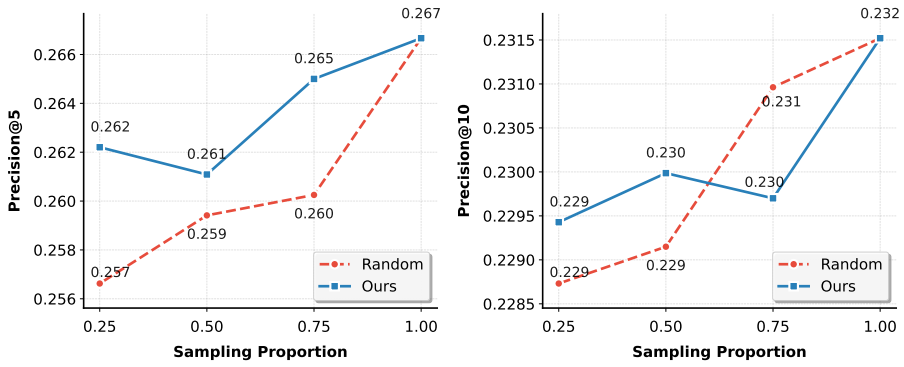


Figure 5.4: Comparison of sampling strategies on ML-1M dataset: random vs. similarity-based.

5.5.3 Efficiency Comparison (RQ1)

Table 5.4 reports training and testing costs. Most LLM baselines are far slower than NLCF due to prompt engineering, sequence augmentation, and online LLM inference. LLMEmb improves inference by shifting to embeddings, but training remains burdened by long natural language graph descriptions and it does not directly model user–item multi-order interactions. In contrast, NLCF is efficient because: (i) training uses concise graph sentences; and (ii) inference uses precomputed collaborative embeddings for full-item retrieval, avoiding runtime generation.

5.5.4 Ablation Study (RQ2)

We analyze backbone size and sampling strategy in Figures 5.3 and 5.4. Since NLCF has a simple architecture with a single training objective, we focus on these two factors. First, performance grows with model size, consistent with empirical scaling laws: 1B already performs well; 7B is stronger; gains from 7B to 8B are smaller, suggesting diminishing returns. Second, our similarity-based sampling outperforms alternatives across most ratios. Random sampling ignores subgraph density and misses representative coverage, whereas similarity-based sampling controls granularity for better sentence grouping, especially beneficial under lower budgets.

5.5.5 Hyper-parameter Sensitivity (RQ3)

We study the effects of sampling ratio α, sentence length l, and similarity threshold t on Table 5.5, and Figure 5.5.

We have several observations: (i) $l = 4$ works best, suggesting third-order collaborative information suffice; longer paths introduce noise. Increasing α improves performance; reducing from 1 to 0.75 yields only a small drop, evidencing effective sampling. (ii) The similarity threshold t is critical: moderate values preserve represen-

Table 5.4: Training and test time among LLM-based models (seconds [s], minutes [m], hours [h]).

Model	**Steam**		**ML-10M**		**ML-1M**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
TransRec	16h5m	3h25m	18h44m	4h18m	4h32m	52m42s
LLM-CF	13h24m	0.029s	16h58m	0.101s	3h47m	0.003s
LETTER	5h6m	3m31s	7h27m	5m35s	1h43m	1m16s
LC-Rec	11h49m	18m59s	13h22m	26m07s	3h18m	7m21s
LLMEmb	41m36s	0.029s	48m15s	0.101s	10m23s	0.003s
NLCF	42m43s	0.029s	44m19s	0.101s	8m12s	0.003s

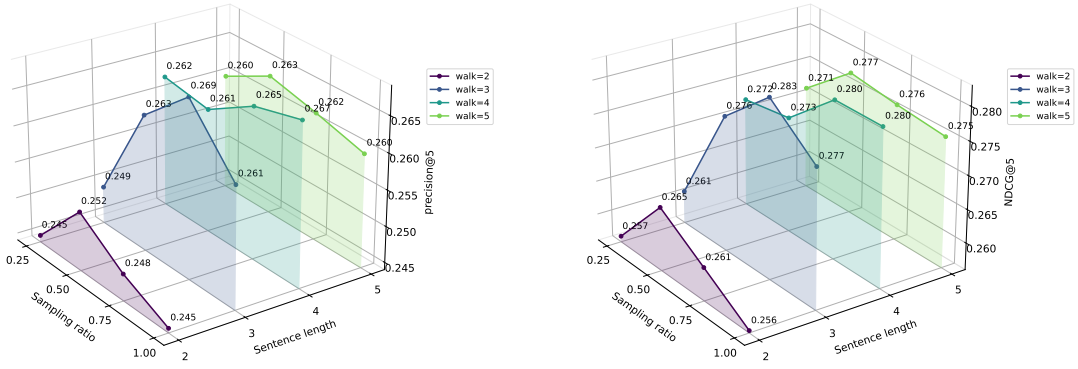
Table 5.5: Effect of similarity threshold t on NLCF performance (ML-1M).

Metric	0.25	0.50	0.75	1.00
Precision@5	0.2421	**0.2650**	0.2575	0.2591
NDCG@5	0.2550	**0.2796**	0.2754	0.2758

tative diversity; overly high t makes sentences appear dissimilar and degenerates to near-random selection; overly low t groups too many sentences together and weakens discrimination.

5.5.6 Online A/B Testing (RQ4)

We deploy NLCF on a Alibaba platform with offline and online centers. Offline, logs are processed to build the graph corpus, train NLCF, and produce user/item embeddings. Embeddings are updated daily by regenerating a corpus from recent interactions; since embeddings are sums of hidden states, incremental updates are straightforward. Online, the service retrieves items using precomputed user embeddings, avoiding real-time LLM reasoning. NLCF replaces LightGCN and LLM-CF in recall. Over eight weeks with 5% traffic per variant, NLCF improves PCTR/UCTR/GMV



(a) Precision@5 (b) NDCG@5

Figure 5.5: Impact of sentence length l and sampling ratio α on Precision@5 and NDCG@5 on ML-1M dataset.

Table 5.6: Online A/B tests on an industrial platform.

A/B Test	**PCTR**	**UCTR**	**GMV**	**ResTime**
v.s. LightGCN	+4.15%	+3.11%	+5.78%	+1.46%
v.s. LLM-CF	+2.51%	+2.47%	+3.14%	-20.17%

over LightGCN by +4.15%/+3.11%/+5.78% with only +1.46% latency. Compared to LLM-CF, NLCF gains +2.51%/+2.47%/+3.14% and reduces latency by 20.17%, highlighting the efficiency of embedding-only serving.

5.6 Summary

We present NLCF, a new paradigm for fine-tuning LLMs on CF by treating the user–item graph as a language. Prior prompt-based methods suffer from: (i) efficiency limits: token-by-token generation in the inference stage; and (ii) topological modeling limits: difficulty encoding multi-order collaboration via natural language descriptions. NLCF is built on two insights: LLMs are strong sequence learners given

suitable corpora, and token transitions can mirror graph transitions. NLCF (i) converts the graph into a corpus capturing multi-order interactions, and (ii) fine-tunes LLMs to internalize these interactions and construct collaborative embeddings for efficient retrieval. Extensive offline results show superior accuracy and markedly better efficiency over LLM baselines; online A/B tests on a large-scale platform confirm effectiveness in practice.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This thesis has presented a step-by-step path towards LLM-based recommender systems, moving the integration from user–item feature to user–item interaction graph. The three papers form a coherent progression that makes LLMs more better understand user-item features, more aware of graph structure, and more practical for real-world recommendation:

- **Self-Monitoring LLMs for CTR Prediction (Paper 1):** This paper shows that LLMs in standard CTR setups often use features in a shallow way, especially on long-tail items, and introduces a self-monitoring framework that better aligns predictions with user–item features for LLMs.
- **Each Graph as a New Language for Graph-Aware LLMs (Paper 2):** This paper treats each graph as a new language, where nodes are tokens and random walks are sentences, and demonstrates that LLMs trained on these graph-defined sequences can efficiently learn high-order graph structure.
- **Efficient LLM-based CF on User–Item Interaction Graphs. (Paper 3):**

This paper models the user–item interaction graph as a collaborative language and trains an LLM on sampled sentences, then aggregates hidden states into user/item embeddings for fast retrieval without online LLM inference.

Collectively, these works advance the use of LLMs in recommendation by going beyond description-based prompting and showing how LLMs can more faithfully use user-item features, directly learn from graph structure, and support scalable retrieval on user–item graphs.

6.2 A Unifying Philosophy

Beyond summarizing individual contributions, looking at FILM, GDL4LLM, and NLCF holistically reveals a unifying philosophy in three dimensions that define the progression of this thesis:

6.2.1 From Generators to Representation Learners

A core dimension of philosophy across all three works is shifting the role of LLMs from open-ended text generators to semantic and structure representation learners. Instead of relying on slow, token-by-token generation for final output, we leverage the LLM's reasoning capabilities to encode complex inputs into dense representations. FILM encodes feature interactions, GDL4LLM encodes attributed graph structures, and NLCF encodes collaborative signals. This representation learning approach ensures that the deep understanding of LLMs is captured in a format compatible with efficient downstream tasks.

6.2.2 Knowledge Injection for Domain Adaptation

A second dimension of the philosophy is addressing the knowledge gap of pre-trained LLMs. While LLMs have rich open-world knowledge, they are inherently blind to domain-specific structural knowledge, e.g., user/item features, interaction graphs, specific node attributes, etc. We propose a unified structural knowledge injection methodology via fine-tuning. FILM injects feature-label relationships; GDL4LLM injects general attributed structure and node label relationships; and NLCF injects user interests via similar user interactions. This demonstrates that successful adaptation requires aligning general semantic reasoning with specific structural data via fine-tuning.

6.2.3 Scalability and Deployment Readiness

The three methods also represent a shared thought in balancing performance with inference efficiency. FILM addresses efficiency via knowledge distillation, transferring LLM insights to lightweight models for fast inference. GDL4LLM improves scalability by optimizing prompt length and encoding graph neighborhoods efficiently. Finally, NLCF achieves the highest deployment readiness by transforming generative recommendations into embedding retrieval. It eliminates online LLM inference latency entirely, making it suitable for billion-scale systems.

6.3 Future Work

While this thesis makes progress in integrating LLMs with recommender systems, several challenges remain open and provide promising directions for future research.

- Dynamic User–Item Graph Understanding: In real-world recommender systems, user–item interactions form a dynamic graph: users continuously interact with

new items, their interests gradually drift, and their preferences show both short-term changes and long-term trends. Directly applying our current framework to such scenarios is challenging, because the underlying graph structure keeps changing. In particular, repeatedly fine-tuning LLMs on the entire dynamic graph whenever it updates is computationally expensive. A natural extension is to design a dynamic version of our framework that can update user and item representations incrementally as new interactions arrive, without re-training on the full history.

- Cold-Start Recommendation: Another important direction is to address the cold-start problem. In the current framework, we introduce new graph tokens and train their representations based on observed interactions, which means the method is mainly suitable for warm users and items. However, in real-world recommender systems there are many cold users and items that have few or even no interactions. For these nodes, the framework has very limited structural information to learn from, which makes it hard to obtain meaningful token embeddings. Future work can explore combining our framework with side information, such as user profiles, item descriptions, or category labels, so that LLMs can build initial representations for cold users and items even before sufficient interactions are observed.

- Toward More Expressive Graph Language Design: The current framework focuses on structural patterns and does not fully make use of language-like properties such as syntax. It treats node IDs as tokens and learns from their co-occurrence patterns in random-walk sequences, which capture connectivity but provide limited other structure information. A promising direction is to design a more expressive graph language that include richer structure signals. For example, one can add grammar rules to introduce edge types, time information, or small subgraph patterns to graph language. These extensions may allow the framework to support more detailed reasoning over graphs.

References

[1] AI@Meta. Llama 3 model card. 2024.

[2] Keqin Bao, Jizhi Zhang, Wenjie Wang, Yang Zhang, Zhengyi Yang, Yancheng Luo, Fuli Feng, Xiangnan He, and Qi Tian. A bi-step grounding paradigm for large language models in recommendation systems. *CoRR*, abs/2308.08434, 2023.

[3] Keqin Bao, Jizhi Zhang, Yang Zhang, Wenjie Wang, Fuli Feng, and Xiangnan He. Tallrec: An effective and efficient tuning framework to align large language model with recommendation. *arXiv preprint arXiv:2305.00447*, 2023.

[4] Xuheng Cai, Chao Huang, Lianghao Xia, and Xubin Ren. Lightgcl: Simple yet effective graph contrastive learning for recommendation. In *The Eleventh International Conference on Learning Representations*, 2023.

[5] Ben Chen, Xian Guo, Siyuan Wang, Zihan Liang, Yue Lv, Yufei Ma, Xinlong Xiao, Bowen Xue, Xuxin Zhang, Ying Yang, et al. Onesearch: A preliminary exploration of the unified end-to-end generative framework for e-commerce search. *arXiv preprint arXiv:2509.03236*, 2025.

[6] Jiajia Chen, Jiancan Wu, Jiawei Chen, Chongming Gao, Yong Li, and Xiang Wang. Position-aware graph transformer for recommendation. *ACM Transactions on Information Systems*, 43(6):1–24, 2025.

[7] Runjin Chen, Tong Zhao, Ajay Jaiswal, Neil Shah, and Zhangyang Wang. Llaga: Large language and graph assistant. *arXiv preprint arXiv:2402.08170*, 2024.

[8] Xu Chen, Zida Cheng, Jiangchao Yao, Chen Ju, Weilin Huang, Jinsong Lan, Xiaoyi Zeng, and Shuai Xiao. Enhancing cross-domain click-through rate prediction via explicit feature augmentation. In Tat-Seng Chua, Chong-Wah Ngo, Roy Ka-Wei Lee, Ravi Kumar, and Hady W. Lauw, editors, *Companion Proceedings of the ACM on Web Conference 2024, WWW 2024, Singapore, Singapore, May 13-17, 2024*, pages 423–432. ACM, 2024.

[9] Zhihong Chen, Jiawei Wu, Chenliang Li, Jingxu Chen, Rong Xiao, and Binqiang Zhao. Co-training disentangled domain adaptation network for leveraging popularity bias in recommenders. In *Proceedings of the 45th International ACM SIGIR conference on research and development in information retrieval*, pages 60–69, 2022.

[10] Zhikai Chen, Haitao Mao, Jingzhe Liu, Yu Song, Bingheng Li, Wei Jin, Bahare Fatemi, Anton Tsitsulin, Bryan Perozzi, Hui Liu, et al. Text-space graph foundation models: Comprehensive benchmarks and new insights. *arXiv preprint arXiv:2406.10727*, 2024.

[11] Kairui Fu, Tao Zhang, Shuwen Xiao, Ziyang Wang, Xinming Zhang, Chenchi Zhang, Yuliang Yan, Junjun Zheng, Yu Li, Zhihong Chen, et al. Forge: Forming semantic identifiers for generative retrieval in industrial datasets. *arXiv preprint arXiv:2509.20904*, 2025.

[12] Jingtong Gao, Zhaocheng Du, Xiaopeng Li, Yichao Wang, Xiangyang Li, Huifeng Guo, Ruiming Tang, and Xiangyu Zhao. Samplellm: Optimizing tabular data synthesis in recommendations. In *Companion Proceedings of the ACM on Web Conference 2025*, pages 211–220, 2025.

[13] Shijie Geng, Shuchang Liu, Zuohui Fu, Yingqiang Ge, and Yongfeng Zhang.

Recommendation as language processing (rlp): A unified pretrain, personalized prompt & predict paradigm (p5). In *Proceedings of the 16th ACM Conference on Recommender Systems*, pages 299–315, 2022.

[14] 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.

[15] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: a factorization-machine based neural network for ctr prediction. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 1725–1731, 2017.

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

[17] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19, 2016.

[18] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 639–648, 2020.

[19] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182, 2017.

[20] Yupeng Hou, Jiacheng Li, Zhankui He, An Yan, Xiusi Chen, and Julian McAuley. Bridging language and items for retrieval and recommendation. *arXiv preprint arXiv:2403.03952*, 2024.

[21] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[22] Guoqing Hu, An Zhang, Shuo Liu, Zhibo Cai, Xun Yang, and Xiang Wang. Alphafuse: Learn id embeddings for sequential recommendation in null space of language embeddings. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1614–1623, 2025.

[23] Jun Hu, Wenwen Xia, Xiaolu Zhang, Chilin Fu, Weichang Wu, Zhaoxin Huan, Ang Li, Zuoli Tang, and Jun Zhou. Enhancing sequential recommendation via llm-based semantic embedding learning. In Tat-Seng Chua, Chong-Wah Ngo, Roy Ka-Wei Lee, Ravi Kumar, and Hady W. Lauw, editors, *Companion Proceedings of the ACM on Web Conference 2024, WWW 2024, Singapore, Singapore, May 13-17, 2024*, pages 103–111. ACM, 2024.

[24] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 2333–2338, 2013.

[25] Xuanwen Huang, Kaiqiao Han, Yang Yang, Dezheng Bao, Quanjin Tao, Ziwei Chai, and Qi Zhu. Can gnn be good adapter for llms? In *Proceedings of the ACM on Web Conference 2024*, pages 893–904, 2024.

[26] Hakan Inan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev, Qing Hu, Brian Fuller, Davide Testuggine, et al. Llama guard: Llm-based input-output safeguard for human-ai conversations. *arXiv preprint arXiv:2312.06674*, 2023.

[27] Yangqin Jiang, Yuhao Yang, Lianghao Xia, Da Luo, Kangyi Lin, and

Chao Huang. Reclm: Recommendation instruction tuning. *arXiv preprint arXiv:2412.19302*, 2024.

[28] Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. Field-aware factorization machines for ctr prediction. In *Proceedings of the 10th ACM conference on recommender systems*, pages 43–50, 2016.

[29] Wang-Cheng Kang and Julian J. McAuley. Self-attentive sequential recommendation. In *IEEE International Conference on Data Mining, ICDM 2018, Singapore, November 17-20, 2018*, pages 197–206. IEEE Computer Society, 2018.

[30] Sein Kim, Hongseok Kang, Seungyoon Choi, Donghyun Kim, Minchul Yang, and Chanyoung Park. Large language models meet collaborative filtering: An efficient all-round llm-based recommender system. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1395–1406, 2024.

[31] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[32] Taeyong Kong, Taeri Kim, Jinsung Jeon, Jeongwhan Choi, Yeon-Chang Lee, Noseong Park, and Sang-Wook Kim. Linear, or non-linear, that is the question! In *Proceedings of the fifteenth ACM international conference on web search and data mining*, pages 517–525, 2022.

[33] Xiaoyu Kong, Leheng Sheng, Junfei Tan, Yuxin Chen, Jiancan Wu, An Zhang, Xiang Wang, and Xiangnan He. Minionerec: An open-source framework for scaling generative recommendation. *arXiv preprint arXiv:2510.24431*, 2025.

[34] Xiaoyu Kong, Jiancan Wu, An Zhang, Leheng Sheng, Hui Lin, Xiang Wang, and Xiangnan He. Customizing language models with instance-wise lora for sequential recommendation. *Advances in Neural Information Processing Systems*, 37:113072–113095, 2024.

[35] Hao Li, Chenghao Yang, An Zhang, Yang Deng, Xiang Wang, and Tat-Seng Chua. Hello again! llm-powered personalized agent for long-term dialogue. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 5259–5276, 2025.

[36] Rui Li, Jiwei Li, Jiawei Han, and Guoyin Wang. Similarity-based neighbor selection for graph llms. *arXiv preprint arXiv:2402.03720*, 2024.

[37] Jiayi Liao, Sihang Li, Zhengyi Yang, Jiancan Wu, Yancheng Yuan, Xiang Wang, and Xiangnan He. Llara: Large language-recommendation assistant. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1785–1795, 2024.

[38] Jiayi Liao, Ruobing Xie, Sihang Li, Xiang Wang, Xingwu Sun, Zhanhui Kang, and Xiangnan He. Multi-grained patch training for efficient llm-based recommendation. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1572–1581, 2025.

[39] Jianghao Lin, Bo Chen, Hangyu Wang, Yunjia Xi, Yanru Qu, Xinyi Dai, Kangning Zhang, Ruiming Tang, Yong Yu, and Weinan Zhang. Clickprompt: CTR models are strong prompt generators for adapting language models to CTR prediction. In Tat-Seng Chua, Chong-Wah Ngo, Ravi Kumar, Hady W. Lauw, and Roy Ka-Wei Lee, editors, *Proceedings of the ACM on Web Conference 2024, WWW 2024, Singapore, May 13-17, 2024*, pages 3319–3330. ACM, 2024.

[40] Jianghao Lin, Bo Chen, Hangyu Wang, Yunjia Xi, Yanru Qu, Xinyi Dai, Kangning Zhang, Ruiming Tang, Yong Yu, and Weinan Zhang. Clickprompt: Ctr models are strong prompt generators for adapting language models to ctr prediction. In *Proceedings of the ACM on Web Conference 2024*, pages 3319–3330, 2024.

[41] Jianghao Lin, Rong Shan, Chenxu Zhu, Kounianhua Du, Bo Chen, Shigang Quan, Ruiming Tang, Yong Yu, and Weinan Zhang. Rella: Retrieval-enhanced large language models for lifelong sequential behavior comprehension in recommendation. In *Proceedings of the ACM on Web Conference 2024*, pages 3497–3508, 2024.

[42] Xinyu Lin, Wenjie Wang, Yongqi Li, Fuli Feng, See-Kiong Ng, and Tat-Seng Chua. Bridging items and language: A transition paradigm for large language model-based recommendation. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1816–1826, 2024.

[43] Xinyu Lin, Wenjie Wang, Yongqi Li, Shuo Yang, Fuli Feng, Yinwei Wei, and Tat-Seng Chua. Data-efficient fine-tuning for llm-based recommendation. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 365–374, 2024.

[44] Zihan Lin, Changxin Tian, Yupeng Hou, and Wayne Xin Zhao. Improving graph collaborative filtering with neighborhood-enriched contrastive learning. In *Proceedings of the ACM web conference 2022*, pages 2320–2329, 2022.

[45] Qidong Liu, Xian Wu, Wanyu Wang, Yejing Wang, Yuanshao Zhu, Xiangyu Zhao, Feng Tian, and Yefeng Zheng. Llmemb: Large language model can be a good embedding generator for sequential recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 12183–12191, 2025.

[46] Qidong Liu, Xian Wu, Yejing Wang, Zijian Zhang, Feng Tian, Yefeng Zheng, and Xiangyu Zhao. Llm-esr: Large language models enhancement for long-tailed sequential recommendation. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[47] Qidong Liu, Xiangyu Zhao, Yuhao Wang, Yejing Wang, Zijian Zhang, Yuqi

Sun, Xiang Li, Maolin Wang, Pengyue Jia, Chong Chen, et al. Large language model enhanced recommender systems: Taxonomy, trend, application and future. *arXiv e-prints*, pages arXiv–2412, 2024.

[48] Xin Liu, Zheng Li, Yifan Gao, Jingfeng Yang, Tianyu Cao, Zhengyang Wang, Bing Yin, and Yangqiu Song. Enhancing user intent capture in session-based recommendation with attribute patterns. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[49] Zipeng Liu, Likang Wu, Ming He, Zhong Guan, Hongke Zhao, and Nan Feng. Dr. e bridges graphs with large language models through words. *arXiv e-prints*, pages arXiv–2406, 2024.

[50] Jianglin Lu, Yixuan Liu, Yitian Zhang, and Yun Fu. Scale-free graph-language models. *arXiv preprint arXiv:2502.15189*, 2025.

[51] Jianglin Lu, Yi Xu, Huan Wang, Yue Bai, and Yun Fu. Latent graph inference with limited supervision. *Advances in Neural Information Processing Systems*, 36:32521–32538, 2023.

[52] Xinchen Luo, Jiangxia Cao, Tianyu Sun, Jinkai Yu, Rui Huang, Wei Yuan, Hezheng Lin, Yichen Zheng, Shiyao Wang, Qigen Hu, et al. Qarm: Quantitative alignment multi-modal recommendation at kuaishou. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, pages 5915–5922, 2025.

[53] Joel Mackenzie, Andrew Trotman, and Jimmy Lin. Efficient document-at-a-time and score-at-a-time query evaluation for learned sparse representations. *ACM Transactions on Information Systems*, 41(4):1–28, 2023.

[54] Kelong Mao, Jieming Zhu, Liangcai Su, Guohao Cai, Yuru Li, and Zhenhua Dong. Finalmlp: An enhanced two-stream mlp model for ctr prediction. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37:4552–4560, 06 2023.

[55] Giannis Nikolentzos, Antoine Tixier, and Michalis Vazirgiannis. Message passing attention networks for document understanding. In *Proceedings of the aaai conference on artificial intelligence*, volume 34, pages 8544–8551, 2020.

[56] OpenAI. Gpt-4 technical report. `https://arxiv.org/abs/2303.08774`, 2023. Accessed: 2023-10-12.

[57] Bo Pan, Zheng Zhang, Yifei Zhang, Yuntong Hu, and Liang Zhao. Distilling large language models for text-attributed graph learning. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pages 1836–1845, 2024.

[58] Haohao Qu, Wenqi Fan, Zihuai Zhao, and Qing Li. Tokenrec: Learning to tokenize id for llm-based generative recommendation. *arXiv preprint arXiv:2406.10450*, 2024.

[59] Yu Song, Haitao Mao, Jiachen Xiao, Jingzhe Liu, Zhikai Chen, Wei Jin, Carl Yang, Jiliang Tang, and Hui Liu. A pure transformer pretraining framework on text-attributed graphs. *arXiv preprint arXiv:2406.13873*, 2024.

[60] Guangxin Su, Hanchen Wang, Jianwei Wang, Wenjie Zhang, Ying Zhang, and Jian Pei. Large language models meet text-attributed graphs: A survey of integration frameworks and applications. *arXiv preprint arXiv:2510.21131*, 2025.

[61] Zhongxiang Sun, Zihua Si, Xiaoxue Zang, Kai Zheng, Yang Song, Xiao Zhang, and Jun Xu. Large language models enhanced collaborative filtering. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pages 2178–2188, 2024.

[62] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 3319–3328. PMLR, 2017.

[63] Jiabin Tang, Yuhao Yang, Wei Wei, Lei Shi, Lixin Su, Suqi Cheng, Dawei Yin, and Chao Huang. Graphgpt: Graph instruction tuning for large language models. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 491–500, 2024.

[64] Zhen Tian, Ting Bai, Wayne Xin Zhao, Ji-Rong Wen, and Zhao Cao. Eulernet: Adaptive feature interaction learning via euler's formula for ctr prediction. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, page 1376–1385, 2023.

[65] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[66] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[67] Dong Wang, Kavé Salamatian, Yunqing Xia, Weiwei Deng, and Qi Zhang. Bert4ctr: An efficient framework to combine pre-trained language model with non-textual features for ctr prediction. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5039–5050, 2023.

[68] Fangye Wang, Hansu Gu, Dongsheng Li, Tun Lu, Peng Zhang, and Ning Gu.

Towards deeper, lighter and interpretable cross network for ctr prediction. pages 2523–2533, 10 2023.

[69] Ruoxi Wang, Rakesh Shivanna, Derek Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed Chi. Dcn v2: Improved deep & cross network and practical lessons for web-scale learning to rank systems. In *Proceedings of the web conference 2021*, pages 1785–1797, 2021.

[70] Wenjie Wang, Honghui Bao, Xinyu Lin, Jizhi Zhang, Yongqi Li, Fuli Feng, See-Kiong Ng, and Tat-Seng Chua. Learnable item tokenization for generative recommendation. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pages 2400–2409, 2024.

[71] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, pages 165–174, 2019.

[72] Yancheng Wang, Ziyan Jiang, Zheng Chen, Fan Yang, Yingxue Zhou, Eunah Cho, Xing Fan, Yanbin Lu, Xiaojiang Huang, and Yingzhen Yang. Recmind: Large language model powered agent for recommendation. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 4351–4364, 2024.

[73] 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. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pages 806–815, 2024.

[74] Zhipeng Wei, Kuo Cai, Junda She, Jie Chen, Minghao Chen, Yang Zeng, Qiang Luo, Wencong Zeng, Ruiming Tang, Kun Gai, et al. Oneloc: Geo-

aware generative recommender systems for local life service. *arXiv preprint arXiv:2508.14646*, 2025.

[75] Jiahao Wu, Qijiong Liu, Hengchang Hu, Wenqi Fan, Shengcai Liu, Qing Li, Xiao-Ming Wu, and Ke Tang. Leveraging large language models (llms) to empower training-free dataset condensation for content-based recommendation. *arXiv preprint arXiv:2310.09874*, 2023.

[76] Junda Wu, Cheng-Chun Chang, Tong Yu, Zhankui He, Jianing Wang, Yupeng Hou, and Julian McAuley. Coral: Collaborative retrieval-augmented large language models improve long-tail recommendation. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3391–3401, 2024.

[77] Likang Wu, Zhaopeng Qiu, Zhi Zheng, Hengshu Zhu, and Enhong Chen. Exploring large language model for graph data understanding in online job recommendations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 9178–9186, 2024.

[78] Likang Wu, Zhi Zheng, Zhaopeng Qiu, Hao Wang, Hongchao Gu, Tingjia Shen, Chuan Qin, Chen Zhu, Hengshu Zhu, Qi Liu, Hui Xiong, and Enhong Chen. A survey on large language models for recommendation. *CoRR*, abs/2305.19860, 2023.

[79] Wei Wu, Chao Wang, Dazhong Shen, Chuan Qin, Liyi Chen, and Hui Xiong. Afdgcf: Adaptive feature de-correlation graph collaborative filtering for recommendations. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1242–1252, 2024.

[80] 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.

[81] Yunjia Xi, Weiwen Liu, Jianghao Lin, Xiaoling Cai, Hong Zhu, Jieming Zhu, Bo Chen, Ruiming Tang, Weinan Zhang, and Yong Yu. Towards open-world recommendation with knowledge augmentation from large language models. In *Proceedings of the 18th ACM Conference on Recommender Systems*, pages 12–22, 2024.

[82] Haotian Xu, Yuning You, and Tengfei Ma. When structure doesn't help: Llms do not read text-attributed graphs as effectively as we expected. In *The Fourth Learning on Graphs Conference*.

[83] Deqing Yang, Yanghua Xiao, Yangqiu Song, and Wei Wang. Semantic-based recommendation across heterogeneous domains. In Charu C. Aggarwal, Zhi-Hua Zhou, Alexander Tuzhilin, Hui Xiong, and Xindong Wu, editors, *2015 IEEE International Conference on Data Mining, ICDM 2015, Atlantic City, NJ, USA, November 14-17, 2015*, pages 1075–1080. IEEE Computer Society, 2015.

[84] Hao Yang, Ziliang Wang, Weijie Bian, and Yifan Zeng. Practice on effectively extracting nlp features for click-through rate prediction. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pages 4887–4893, 2023.

[85] Hao Yang, Ziliang Wang, Weijie Bian, and Yifan Zeng. Practice on effectively extracting nlp features for click-through rate prediction. pages 4887–4893, 10 2023.

[86] Junhan Yang, Zheng Liu, Shitao Xiao, Chaozhuo Li, Defu Lian, Sanjay Agrawal, Amit Singh, Guangzhong Sun, and Xing Xie. Graphformers: Gnn-nested transformers for representation learning on textual graph. *Advances in Neural Information Processing Systems*, 34:28798–28810, 2021.

[87] Yanwu Yang and Panyu Zhai. Click-through rate prediction in online advertis-

ing: A literature review. *Information Processing & Management*, 59(2):102853, 2022.

[88] Ruosong Ye, Caiqi Zhang, Runhui Wang, Shuyuan Xu, and Yongfeng Zhang. Language is all a graph needs. In *Findings of the Association for Computational Linguistics: EACL 2024*, pages 1955–1973, 2024.

[89] Wencai Ye, Mingjie Sun, Shuhang Chen, Wenjin Wu, and Peng Jiang. Align ̃3 gr: Unified multi-level alignment for llm-based generative recommendation. *arXiv preprint arXiv:2511.11255*, 2025.

[90] Wencai Ye, Mingjie Sun, Shaoyun Shi, Peng Wang, Wenjin Wu, and Peng Jiang. Das: Dual-aligned semantic ids empowered industrial recommender system. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, pages 6217–6224, 2025.

[91] Sho Yokoi, Ryo Takahashi, Reina Akama, Jun Suzuki, and Kentaro Inui. Word rotator's distance. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2944–2960, 2020.

[92] Runlong Yu, Yuyang Ye, Qi Liu, Zihan Wang, Chunfeng Yang, Yucheng Hu, and Enhong Chen. Xcrossnet: Feature structure-oriented learning for click-through rate prediction. In Kamal Karlapalem, Hong Cheng, Naren Ramakrishnan, R. K. Agrawal, P. Krishna Reddy, Jaideep Srivastava, and Tanmoy Chakraborty, editors, *Advances in Knowledge Discovery and Data Mining - 25th Pacific-Asia Conference, PAKDD 2021, Virtual Event, May 11-14, 2021, Proceedings, Part II*, volume 12713 of *Lecture Notes in Computer Science*, pages 436–447. Springer, 2021.

[93] Yaodong Yu, Kwan Ho Ryan Chan, Chong You, Chaobing Song, and Yi Ma. Learning diverse and discriminative representations via the principle of maximal

coding rate reduction. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[94] 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. In Hsin-Hsi Chen, Wei-Jou (Edward) Duh, Hen-Hsen Huang, Makoto P. Kato, Josiane Mothe, and Barbara Poblete, editors, *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2023, Taipei, Taiwan, July 23-27, 2023*, pages 2639–2649. ACM, 2023.

[95] Zhenrui Yue, Sara Rabhi, Gabriel de Souza Pereira Moreira, Dong Wang, and Even Oldridge. Llamarec: Two-stage recommendation using large language models for ranking. *CoRR*, abs/2311.02089, 2023.

[96] An Zhang, Yuxin Chen, Leheng Sheng, Xiang Wang, and Tat-Seng Chua. On generative agents in recommendation. In *Proceedings of the 47th international ACM SIGIR conference on research and development in Information Retrieval*, pages 1807–1817, 2024.

[97] Chao Zhang, Haoxin Zhang, Shiwei Wu, Di Wu, Tong Xu, Xiangyu Zhao, Yan Gao, Yao Hu, and Enhong Chen. Notellm-2: Multimodal large representation models for recommendation. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pages 2815–2826, 2025.

[98] Jun Zhang, Yi Li, Yue Liu, Changping Wang, Yuan Wang, Yuling Xiong, Xun Liu, Haiyang Wu, Qian Li, Enming Zhang, et al. Gpr: Towards a generative pre-trained one-model paradigm for large-scale advertising recommendation. *arXiv preprint arXiv:2511.10138*, 2025.

[99] Qinggang Zhang, Shengyuan Chen, Yuanchen Bei, Zheng Yuan, Huachi Zhou, Zijin Hong, Hao Chen, Yilin Xiao, Chuang Zhou, Junnan Dong, et al. A survey of graph retrieval-augmented generation for customized large language models. *arXiv preprint arXiv:2501.13958*, 2025.

[100] Yang Zhang, Keqin Bao, Ming Yan, Wenjie Wang, Fuli Feng, and Xiangnan He. Text-like encoding of collaborative information in large language models for recommendation. *arXiv preprint arXiv:2406.03210*, 2024.

[101] Yang Zhang, Fuli Feng, Jizhi Zhang, Keqin Bao, Qifan Wang, and Xiangnan He. Collm: Integrating collaborative embeddings into large language models for recommendation. *arXiv preprint arXiv:2310.19488*, 2023.

[102] Yifei Zhang, Hao Zhu, Zixing Song, Piotr Koniusz, and Irwin King. Costa: covariance-preserving feature augmentation for graph contrastive learning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2524–2534, 2022.

[103] Zhaoqi Zhang, Haolei Pei, Jun Guo, Tianyu Wang, Yufei Feng, Hui Sun, Shaowei Liu, and Aixin Sun. Onetrans: Unified feature interaction and sequence modeling with one transformer in industrial recommender. *arXiv preprint arXiv:2510.26104*, 2025.

[104] Zijian Zhang, Shuchang Liu, Ziru Liu, Rui Zhong, Qingpeng Cai, Xiangyu Zhao, Chunxu Zhang, Qidong Liu, and Peng Jiang. Llm-powered user simulator for recommender system. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 13339–13347, 2025.

[105] Huan Zhao, Quanming Yao, Jianda Li, Yangqiu Song, and Dik Lun Lee. Meta-graph based recommendation fusion over heterogeneous information networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowl-*

edge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017, pages 635–644. ACM, 2017.

[106] Jianan Zhao, Meng Qu, Chaozhuo Li, Hao Yan, Qian Liu, Rui Li, Xing Xie, and Jian Tang. Learning on large-scale text-attributed graphs via variational inference. *arXiv preprint arXiv:2210.14709*, 2022.

[107] Bowen Zheng, Yupeng Hou, Hongyu Lu, Yu Chen, Wayne Xin Zhao, Ming Chen, and Ji-Rong Wen. Adapting large language moda survey of large language modelsels by integrating collaborative semantics for recommendation. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pages 1435–1448. IEEE, 2024.

[108] Tianshi Zheng, Zheye Deng, Hong Ting Tsang, Weiqi Wang, Jiaxin Bai, Zihao Wang, and Yangqiu Song. From automation to autonomy: A survey on large language models in scientific discovery. *CoRR*, abs/2505.13259, 2025.

[109] Guorui Zhou, Jiaxin Deng, Jinghao Zhang, Kuo Cai, Lejian Ren, Qiang Luo, Qianqian Wang, Qigen Hu, Rui Huang, Shiyao Wang, et al. Onerec technical report. *arXiv preprint arXiv:2506.13695*, 2025.

[110] Huachi Zhou, Hao Chen, Junnan Dong, Daochen Zha, Chuang Zhou, and Xiao Huang. Adaptive popularity debiasing aggregator for graph collaborative filtering. In *Proceedings of the 46th international ACM SIGIR conference on research and development in information retrieval*, pages 7–17, 2023.

[111] Huachi Zhou, Jiahe Du, Chuang Zhou, Chang Yang, Yilin Xiao, Yuxuan Xie, and Xiao Huang. Each graph is a new language: Graph learning with llms. *arXiv preprint arXiv:2501.11478*, 2025.

[112] Huachi Zhou, Jiaqi Fan, Xiao Huang, Ka Ho Li, Zhenyu Tang, and Dahai Yu. Multi-interest refinement by collaborative attributes modeling for click-through

rate prediction. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 4732–4736, 2022.

[113] Huachi Zhou, Kaijing Yu, Qinggang Zhang, Hao Chen, Daochen Zha, Wenqi Pei, Anthony Kong, and Xiao Huang. Self-monitoring large language models for click-through rate prediction. *ACM Transactions on Information Systems*, 44(1):1–25, 2025.

[114] Huachi Zhou, Yujing Zhang, Hao Chen, Qinggang Zhang, Qijie Shen, Feiran Huang, and Xiao Huang. Llm collaborative filtering: User-item graph as new language. In *The Fortieth AAAI Conference on Artificial Intelligence*, 2026.

[115] Luyao Zhuang, Shengyuan Chen, Yilin Xiao, Huachi Zhou, Yujing Zhang, Hao Chen, Qinggang Zhang, and Xiao Huang. Linearrag: Linear graph retrieval augmented generation on large-scale corpora. *arXiv preprint arXiv:2510.10114*, 2025.

[116] Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In Allan Ellis and Tatsuya Hagino, editors, *Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005*, pages 22–32. ACM, 2005.