LLM Collaborative Filtering:
User-Item Graph as New Language

Huachi Zhou[1], **Yujing Zhang**[1], **Hao Chen**[2*],
Qinggang Zhang[1], **Qijie Shen**[3], **Feiran Huang**[4], **Xiao Huang**[1]
[1]The Hong Kong Polytechnic University, Hong Kong
[2]City University of Macau, Macao
[3]Alibaba Group, China
[4]Jinan University, China
{huachi.zhou, yu-jing.zhang, qinggangg.zhang}@connect.polyu.hk
sundaychenhao@gmail.com, qjshenxdu@gmail.com
huangfr@jnu.edu.cn, xiaohuang@comp.polyu.edu.hk

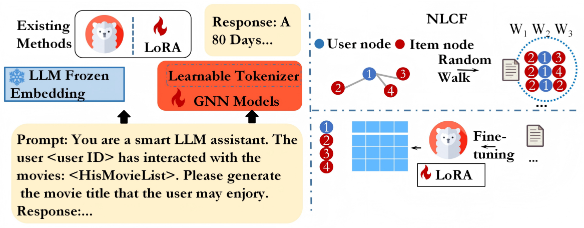
Abstract

In collaborative filtering, learning effective embeddings for users and items from interaction data remains a central challenge. While recent efforts leverage large language models (LLMs) to enhance collaborative filtering, two critical limitations persist: (1) Efficiency: LLM-based inference is significantly slower than traditional embedding-based search; and (2) Topological Modeling: LLMs struggle to capture graph structures, which are essential for modeling multi-order user-item interactions. To address these limitations, we propose New Language Collaborative Filtering (NLCF), a framework that aligns LLMs with collaborative filtering by conceptualizing user-item graphs as new languages. This approach is based on two key insights: (1) LLMs excel at mastering new languages when trained on suitable corpora, and (2) the empirical conditional probability between tokens in corpora converges to the transition probabilities between nodes in graphs. NLCF translates user-item graphs into corpora, where users and items are treated as tokens. These corpora are used to fine-tune LLMs, and the learned representations are aggregated to construct user and item embeddings that encode multi-order interactions. Unlike methods that deploy LLMs for inference, NLCF distills LLM knowledge learned from corpora into compact embeddings, enabling both efficient training and real-time inference. The framework has been deployed on a billion-scale e-commerce platform for several months. Extensive experiments demonstrate that NLCF outperforms traditional graph CF models and LLM-based baselines while achieving significant training and inference efficiency improvement over LLM-based baselines.

Introduction

Collaborative Filtering (CF) plays a pivotal role in recommender systems by suggesting new items to users based on the collaborative information that users with similar interactions share similar preferences (Yuan et al. 2023). Multi-order interactions which represent the multi-order relationships connecting users and items through various paths in the user-item graph enrich this information by uncovering

[*]Corresponding author.



Prompt: You are a smart LLM assistant. The user <user ID> has interacted with the movies: <HisMovieList>. Please generate the movie title that the user may enjoy. Response:...

Figure 1: Comparison of pipelines between existing methods and NLCF. NLCF achieves efficiency through using short graph sentences for training and embeddings for inference.

preferences beyond immediate interactions. To model multi-order interactions, traditional models like LightGCN (He et al. 2020) achieve this by iteratively aggregating embeddings from immediate interactions in the user-item graphs to learn user and item embeddings. Recently, the capabilities of large language models (LLMs) (Chen et al. 2024; Hong et al. 2024) have inspired their application in CF. However, aligning LLMs with CF tasks remains challenging, as LLMs are optimized for natural language tasks rather than learning user and item embeddings based on user-item graphs.

Existing methods attempt to adapt LLMs for CF by transforming a user's immediate interactions into natural language graph descriptions. These approaches can be broadly categorized into two classes: (i) **Description Reasoning-based** methods: These methods leverage LLMs' natural language understanding to reason about collaborative information. For instance, TransRec (Lin et al. 2024b) constructs descriptive sequences from item identifiers, while LLM-CF (Sun et al. 2024) employs Chain-of-Thought prompting to distill collaborative information explicitly. (ii) **Embedding Injection-based** methods: Methods like LC-Rec (Zheng et al. 2024) and LETTER (Wang et al. 2024) inject user and item embeddings from traditional models (e.g., LightGCN) into prompts. By tokenizing collaborative embeddings and combining them with natural language descriptions, these methods aim to capture both semantic and collaborative information.

Despite their potential, natural language graph description-based methods face one or both of the following limitations in efficiency and topological modeling for CF tasks.

(1) **Multi-order Collaborative Information Loss**: LLMs can process only a limited number of interactions at a time, making it extremely expensive to model multi-order interactions and the underlying collaborative information during training. While some methods inject collaborative embeddings into prompts (Zheng et al. 2024; Wang et al. 2024), fine-tuning LLMs on precomputed embeddings, rather than the original user-item graph, inevitably leads to information loss (Yang et al. 2024).

(2) **Slow Inference**: LLMs generate items token by token, with each token depending on the entire prompt (Zhao et al. 2023). The computational cost of generation scales with the prompt length. This sequential generation process, especially when using lengthy graph description prompts, significantly slows down inference compared to embedding-based search.

Although recent methods, e.g., LLMEmb (Liu et al. 2025) and LLM-CF (Sun et al. 2024), have proposed efficient inference mechanisms, they still focus on generating accurate user and item profile representations rather than modeling the user-item graph with LLMs. Given that user-item graphs have been proven highly effective for capturing multi-order collaborative information (Wei et al. 2024), modeling the graph with LLMs powerful learning ability is a promising direction. However, LLMs excel at learning a new language but find it challenging to efficiently learn from the graph with two key reasons:

(1) **Structural Mismatch between Graph and LLM Inputs**: User-item graphs are irregular, two-dimensional structures, while LLMs are designed to process one-dimensional sequential data. Unlike graph CF models, which aggregate multi-order interactions through graph structure (Wu et al. 2020), LLMs lack a built-in mechanism to handle such graph structures.

(2) **Computational Constraints**: As the order increases, the number of multi-order interactions grows geometrically. Modeling these interactions while maintaining a concise representation is necessary since LLMs, containing billions of parameters, are computationally expensive during both fine-tuning and inference. And deploying LLMs in real-world recommender systems is particularly challenging due to latency constraints.

To address these challenges, we propose New Language Collaborative Filtering (NLCF), a novel framework that enables LLMs to efficiently learn collaborative embeddings by treating the graph as a new language. As shown in Figure 1, NLCF transforms the graph into concise corpus, where empirical conditional probabilities between tokens in the corpus converge to the transition probabilities between nodes in the graph. This transformation enables NLCF to model multi-order interactions, and then NLCF balances common and rare interactions in the corpus through similarity-based sampling. The resulting graph corpus is then used to fine-tune the LLM efficiently, allowing it to construct collaborative embeddings. NLCF achieves efficient training by using compact graph corpus rather than lengthy natural language graph descriptions, and efficient inference by leveraging collaborative embeddings instead of token-by-token generation. Our contributions are as follows:

- We propose NLCF, a novel framework that efficiently integrates LLMs with CF tasks to learn user and item collaborative embeddings by treating the user-item graph as a new language.
- We design two core modules: (i) a graph corpus collection module that transforms the graph into concise corpus, modeling multi-order interactions; and (ii) a collaborative embedding construction module that fine-tunes LLMs on this corpus to construct collaborative embeddings for efficient item search.
- Extensive experiments on three datasets show performance gains over both traditional graph CF and LLM-based baselines, with significant efficiency improvement over LLM-based baselines. Online A/B tests further validate NLCF's effectiveness in industrial applications.

Preliminary

Notation. We represent the user-item graph as a tuple $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \mathcal{U} \cup \mathcal{I}$ denotes the union of user nodes $\mathcal{U}$ and item nodes $\mathcal{I}$. The node set is indexed as $\{v_1, v_2, \ldots, v_{|\mathcal{V}|}\}$, where $|\mathcal{V}|$ is the total number of nodes. The edge set $\mathcal{E} \subseteq \mathcal{U} \times \mathcal{I}$ represents user-item interactions, with cardinality $|\mathcal{E}|$. These interactions are encoded in the adjacency matrix $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$:

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

The bipartite structure ensures $\mathbf{A}_{ui} = 0$ for all user-user and item-item pairs. We fine-tune LLMs on user-item graph $\mathcal{G}$ with LoRA (Hu et al. 2021) $\hat{\mathbf{W}}$ to learn user embeddings $\mathbf{h}_{v_u}$ and item embeddings $\mathbf{h}_{v_i}$. More preliminary details are put in Appendix D.

New Language Collaborative Filtering

In this section, we present NLCF, an efficient framework that applies LLMs to learn collaborative embeddings from the user-item graph through new language learning. As shown in Figure 2, NLCF consists of two primary modules: (i) **Graph Corpus Collection:** The user-item graph is transformed into a new language corpus that encodes multi-order interactions. We retrieve this corpus by employing similarity-based sampling to reduce the computational burden. (ii) **Collaborative Embedding Construction:** The retrieved corpus is used to fine-tune LLMs, enabling the model to capture multi-order collaborative information. Hidden representations from the fine-tuned model are then aggregated to construct collaborative user and item embeddings for inference.

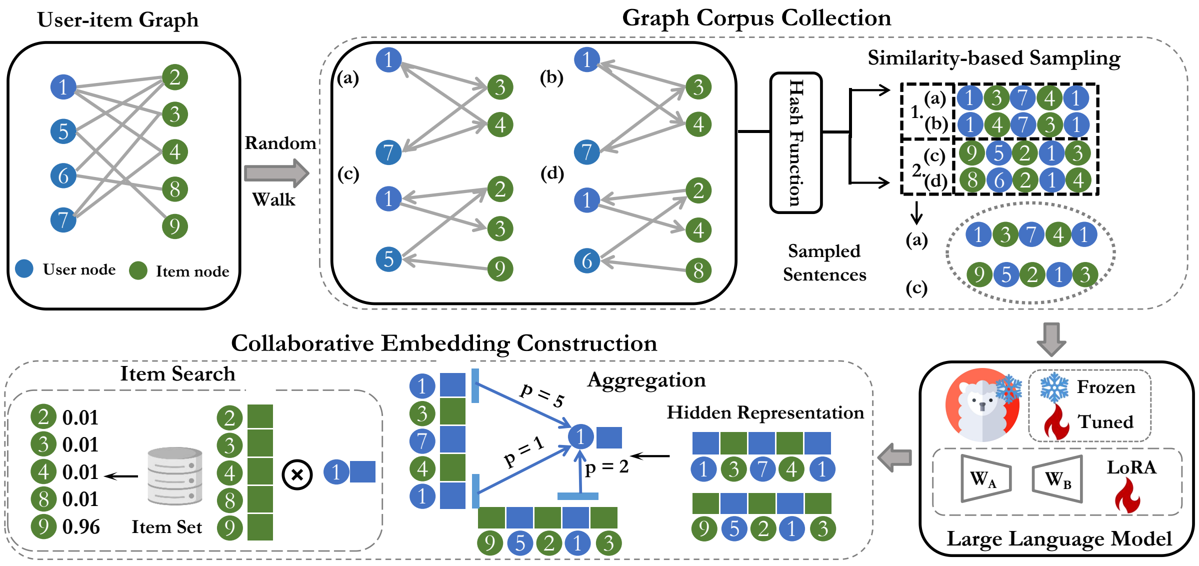


Figure 2: The overall pipeline of the proposed NLCF framework. NLCF treats the user-item graph as a new language. The framework consists of two key modules: in the top part, NLCF first collects graph corpus through random walks and then applies similarity-based sampling; in the bottom part, NLCF constructs collaborative user-item embeddings using fine-tuned LLMs for efficient item search.

Graph Corpus Collection

We begin by mapping basic concepts in user-item graph to their counterparts in graph language and then describe our similarity-based sampling strategy for reducing corpus size.

Definition of Graph Language-related Concepts. User and Item Nodes as Graph Tokens. We define the union of user and item nodes $\mathcal{V}$ as a set of unique graph tokens in this new language and extend the LLM tokenizer's vocabulary accordingly. These newly added tokens have randomly initialized embeddings.

Graph Path as Graph Sentence. A connected sequence of nodes in the graph, or a path, forms a graph sentence in the corpus. For example, as shown in Figure 2, the path $\mathbf{s}_1 = \{v_9, v_5, v_2, v_1, v_3\}$ represents a graph sentence. These paths model multi-order interactions and capture collaborative information by revealing behavior similarity. For instance, v_1 is likely to interact with v_9 because the second-order neighbor user v_5 has previously interacted with v_9. The initial corpus $\mathcal{S} = \{\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, \dots\}$ is composed of such sentences extracted from the user-item graph.

To extract graph sentences from the user-item graph, we employ random walks (Grover and Leskovec 2016), whose transition probability between nodes is defined as:

$$P_g(v_i \mid v_u) = \begin{cases} \frac{\mathbf{A}_{ui}}{\sum_{j \in \mathcal{N}(v_u)} \mathbf{A}_{uj}}, & v_i \in \mathcal{N}(v_u), \\ 0, & (v_u, v_i) \notin \mathcal{E}, \end{cases} \quad (2)$$

where $\mathcal{N}(v_u)$ denotes the neighbor set of node v_u, and $\mathbf{A}_{ui}$ represents the edge weight between nodes v_u and v_i. Each random walk continues walking until reaching a predefined length l. This process is designed to ensure that empirical conditional probability between tokens in resulting corpus converges to the transition probabilities between nodes in graphs. The guarantee is proved by the following theorem:

Theorem 1. *Let $P_s(v_i|v_u)$ be the empirical conditional probability computed from the corpus generated by a sufficiently large number of random walks. As the number of walks approaches infinity, $P_s(v_i|v_u)$ converges in probability to $P_g(v_i|v_u)$.*

The proof is provided in Appendix A. To sufficiently capture collaborative information, we generate d_u sequences starting at each node v_u, where $d_u = \sum \mathbf{A}_{u:}$ is the degree of v_u. This strategy yields the initial corpus $\mathcal{S}$, which serves as the foundation for the following processing.

Similarity-based Sampling for Graph Corpus Reduction. The scale of possible graph corpus is proportional to the number of nodes and the average node degree. To limit the corpus size, traditional method, e.g., random sampling or node degree-based sampling does not account for subgraph density around each node. In dense subgraphs, random walks tend to generate highly overlapping sentences due to frequent visits to common nodes. Some existing LLM-based recommendation sampling methods (Lin et al. 2024a,c) focus on selecting important interactions for a small set of candidate items, which are incompatible with our goal of efficient item search from the entire item set (Zhou et al. 2025d). Other approaches rely on LLM fine-tuning (Zhou et al. 2025c) or LLM data integration to identify important samples (Wu et al. 2023) which would be computationally expensive to use.

We propose a similarity-based sampling strategy that addresses these limitations. Our approach groups similar graph sentences into clusters based on their sentence overlapping and assigns lower sampling probabilities to densely populated clusters. By sampling representative sentences within a pre-defined budget, this strategy reduces redundancy while preserving essential collaborative information.

We measure the overlapping between any two graph sentences using the Jaccard similarity:

$$S_{\text{similarity}}(\mathbf{s}_p, \mathbf{s}_q) = \frac{|\mathbf{s}_p \cap \mathbf{s}_q|}{|\mathbf{s}_p \cup \mathbf{s}_q|}, \tag{3}$$

where $\mathbf{s}_p$ and $\mathbf{s}_q$ represent two graph sentences. We then group sentences into clusters based on their pairwise similarities. A cluster $\mathcal{C}_i$ is defined as a set of sentences where each sentence shares a similarity above threshold t with all other sentences in the cluster:

$$\mathcal{C}_i = \mathbf{s}_p \in \mathcal{S} : S_{\text{similarity}}(\mathbf{s}_p, \mathbf{s}_q) \geq t, \forall \mathbf{s}_q \in \mathcal{C}_i. \tag{4}$$

The size of each cluster $|\mathcal{C}_i|$ reflects the density of similar sentences within it. To achieve balanced representation, we assign higher sampling probabilities to sentences from larger clusters. The sampling probability for each sentence is normalized across all clusters:

$$P(\mathbf{s}_q) = \frac{|\mathcal{C}_i|}{\sum_{j=1}^{m} |\mathcal{C}_j|} : \mathbf{s}_q \in \mathcal{C}_i, \tag{5}$$

where m is the cluster number automatically determined by the algorithm. Then given a pre-defined sampling ratio $\alpha \in (0, 1]$, we sample sentences according to these normalized probabilities to create a representative subset:

$$\mathcal{S}' = \mathbf{s}_q \sim P(\mathbf{s}_q) : \mathbf{s}_q \in \mathcal{S}, |\mathcal{S}'| = \alpha|\mathcal{S}|. \tag{6}$$

This stratified sampling approach ensures balanced representation across clusters while maintaining computational efficiency within the specified budget constraints.

While computing pairwise similarities using Eq. (3) provides fine-grained clustering, it introduces a significant computational complexity of $O(|S|^2)$, which is impractical for large corpus. To address this challenge, we employ MinHash to efficiently estimate Jaccard similarities, whose efficiency is guaranteed by the following theorem:

Theorem 2. *Let $J(s_p, s_q)$ be the Jaccard similarity between two graph sentences s_p and s_q. The MinHash estimator, $\hat{J}(s_p, s_q)$, constructed from k independent hash functions, is an unbiased estimator of $J(s_p, s_q)$ with a variance of $\frac{J(s_p,s_q)(1-J(s_p,s_q))}{k}$.*

The proof is detailed in the Appendix B. This theorem demonstrates that MinHash provides a statistically sound approximation of the true Jaccard similarity. By generating compact signatures for each sentence and storing them in hash tables, we can identify similar sentences with an expected complexity of $\mathcal{O}(|\mathcal{S}|)$, thus making the similarity estimation feasible for large-scale graph corpora.

Theoretical Analyses about Connection between Graph Corpus and Collaborative Information. The graph corpus collection module models multi-order interactions by transforming the two-dimensional user-item graph into a one-dimensional graph corpus. It is important to formally analyze whether this transformation quantitatively guarantees the preservation of multi-order collaborative information in the graph corpus. To address this, we present the following theorem:

Theorem 3. *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 (Hamilton, Ying, and Leskovec 2017), defined as $\text{ReLU}\left(\frac{1}{d_u} \cdot \sum_{v_j \in \mathcal{N}(v_u)} \mathbf{W}_l h_{v_j}^{(l-1)}\right)$. 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. (2) 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.

Collaborative Embedding Construction

Having transformed the user-item graph into a new language corpus rich in collaborative information, we proceed to fine-tune LLMs to incorporate this information. After fine-tuning, to enable efficient inference across the entire item set, we construct collaborative user and item embeddings by aggregating the hidden representations derived from the curated corpus.

Fine-tuning LLMs on the Corpus. Within these graph sentences, multi-order collaborative information enables distant tokens to influence the prediction of the next token in the sentence. To capture this collaborative information, we fine-tune LLMs by maximizing the likelihood of predicting the next token within the graph corpus. Formally, the fine-tuning objective is defined as:

$$\mathcal{L}_{\text{pre}} = -\sum_{q=1}^{|\mathcal{S}'|}\sum_{j=1}^{|\mathbf{s}_q|} \log P(s_{q,j} \mid \mathbf{s}_{q,<j}, \mathbf{W}_p, \hat{\mathbf{W}}), \tag{7}$$

where $\mathbf{W}_p \in \mathbb{R}^{|V|\times d}$ is the learnable head layer parameter for predicting the next token. By optimizing this objective, the fine-tuned LLM learns to capture multi-order collaborative information embedded in the graph sentences. To control the memory usage, we incorporate the dynamic memory bank mechanism and the details are put in the Appendix E.

Graph Sentence Representation Aggregation. After fine-tuning, we obtain a well-trained LLM. However, directly deploying LLMs in an online recommender system would incur prohibitive latency. To address this issue, we precompute collaborative user and item embeddings offline, enabling efficient recommendation at inference time.

LLM is used to compute hidden representations for each graph token in the vocabulary:

$$\mathbf{h}_{q,j} = \text{LLM}(\{s_{q,1}, s_{q,2}, \ldots, s_{q,j-1}\}), \tag{8}$$

where $\mathbf{h}_{q,j}$ represents the hidden representation of the j-th token in sentence $\mathbf{s}_q$, computed based on its preceding tokens $s_{q,1}, s_{q,2}, \ldots, s_{q,j-1}$.

To comprehensively encode multi-order collaborative information, we aggregate the hidden representations of the target user v_u across multiple graph sentences. The aggregation is defined as:

$$\begin{gathered}\mathbf{h}_{v_u} = \sum_{p\in\mathcal{K}} \mathbf{h}_{q,p}, \\ \mathcal{K} = \{p : \mathbf{s}_q \in \mathcal{S}',\ s_{q,p} = v_u,\ p = |\mathbf{s}_q| - k,\ 0 < k < |\mathbf{s}_q|\}\end{gathered} \tag{9}$$

where $\mathcal{K}$ specifies the valid positions of the token v_u across the sampled corpus $\mathcal{S}'$, and k is a hyperparameter controlling the allowed positions of user v_u within a sentence. The item embeddings are computed analogously through the same aggregation process.

By precomputing user and item embeddings offline, NLCF enables efficient real-time inference through simple inner product search with these embeddings, eliminating the need for costly LLM computations during serving.

Experiments

We conduct extensive experiments on three real-world datasets to evaluate the effectiveness and efficiency of the NLCF framework. Our experimental study aims to address the following research questions: **RQ1**: How does NLCF perform compared to state-of-the-art graph CF and LLM-based baselines? **RQ2**: How do different design choices affect NLCF's performance, particularly regarding sampling strategies and LLM backbone selections? **RQ3**: How sensitive is NLCF to key hyperparameters, such as sampling ratio and sentence length? **RQ4**: How does NLCF perform in real-world recommendation applications?

Experimental Settings

Datasets. We evaluate NLCF on three real-world datasets: Steam (Kang and McAuley 2018), ML-1M and ML-10M (Harper and Konstan 2016). Details about these datasets are shown in Appendix F.1. Specifically, Steam contains 918,951 interactions, 41,008 users, and 2,438 items. ML-10M contains 2,340,369 interactions, 69,428 users, and 5,180 items. ML-1M contains 370,647 interactions, 4,869 users, and 1,818 items.

Baseline Methods. We compare NLCF with the following groups of baselines: Traditional Graph CF Baselines: (i) LightGCN (He et al. 2020),(ii) LightGCL (Cai et al. 2023),(iii) HMLET (Kong et al. 2022), and (iv) AFDGCF (Wu et al. 2024b); LLM-based Baselines: TransRec (Lin et al. 2024b), LLM-CF (Sun et al. 2024), LETTER (Wang et al. 2024), LC-Rec (Zheng et al. 2024) and LLMEmb (Liu et al. 2025). Details about these baselines are shown in Appendix F.2. To evaluate the effectiveness of the similarity-based sampling approach, we compare it with random sampling.

Implementation Details. All experiments are conducted using publicly released codes, with each baseline running on **one dedicated** NVIDIA A100-SXM4-40GB GPU. The software environment is based on 20.04.6. The Python version is 3.9.22. We use the Hugging Face Transformers library 4.45.2. And we use metric Precision@N (Zhuang et al. 2025; Zhang et al. 2025) and NDCG@N (Zhou et al. 2023). More implementation details are shown in Appendix F.3.

Main Comparison (RQ1)

Tables 1 and 4 present the overall comparison across three datasets, revealing two key observations:

First, LLM-based approaches do not consistently achieve performance improvement over traditional graph CF approaches across all metrics. LLM-based approaches face challenges in effectively incorporating collaborative information from user-item graph. While methods like LC-Rec and LETTER attempt to integrate collaborative embeddings from traditional graph CF models, these methods that rely on intermediate embeddings instead of directly modeling multi-order interactions suffer from collaborative information loss. This limitation may explain their performance degradation as metric k increases and their inability to consistently outperform state-of-the-art graph CF models, particularly in Precision metrics.

Second, NLCF achieves the most superior performance across all three datasets, demonstrating a successful paradigm shift. Rather than relying on traditional graph CF methods to model multi-order interactions, NLCF enables LLMs to directly capture the collaborative information from the user-item graph, marking a paradigm shift.

Efficiency Comparison (RQ1)

Table 2 presents the training and testing efficiency results across three datasets, revealing that most LLM-based baselines exhibit significantly longer training and inference times compared to NLCF. This performance gap is expected, as LLM-specific strategies—such as user sequence augmentation and diverse prompt template designs—add considerable computational overhead. During inference, many LLM-based approaches still rely on active LLM computations, further increasing inference costs. While LLMEmb fine-tunes LLMs to generate side information from user-item interactions, it struggles with the inefficiency of lengthy natural language graph descriptions during training. As a result, it only improves inference efficiency compared to earlier methods. Notably, LLMEmb does not model user-item interactions directly; instead, it fine-tunes LLMs based on user-item attributes, which fails to encode multi-order user-item interactions and limits potential performance gains over other LLM-based approaches. In contrast, NLCF achieves remarkable efficiency through two key design choices: (i) during training, it fine-tunes LLMs on concise graph sentences derived from the user-item graph; and (ii) during inference, it constructs collaborative embeddings, enabling efficient item retrieval across the entire item set. These designs significantly reduce both training and inference computational costs, making NLCF more efficient than most LLM-based baselines.

Ablation Study (RQ2)

We examine different variants of NLCF through experiments shown in Figure 4 and 6. Since NLCF employs a straightforward architecture without composite components or multiple training objectives, we focus our analysis on two key factors beyond hyper-parameters: the LLM backbone and sampling strategy. Our experiments reveal two significant observations:

First, NLCF's performance aligns with empirical neural scaling laws. As the LLM parameter size increases, the model's performance improves substantially. For instance, the Llama 1B model performs well, highlighting the power of LLMs, while the 7B model achieves better results. However, the performance improvement from 7B to 8B models

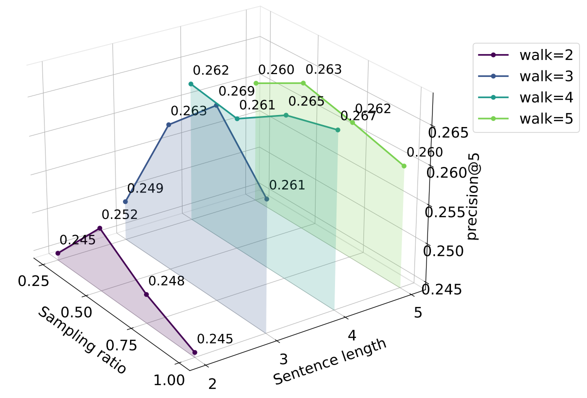


Figure 3: The impact of graph sentence length l and sampling ratio α on Precision@5 on ML-1M dataset.

is relatively modest, suggesting a saturation point in model scaling benefits.

Second, our comparison with alternative sampling strategies demonstrates the superiority of our similarity-based approach across almost all sampling ratios. This advantage likely stems from two factors: random sampling fails to account for subgraph density and struggles to preserve representative samples from each subgraph, while our similarity-based sampling controls the granularity needed for effective graph sentence grouping, particularly at lower sampling ratios where performance gap becomes significant.

Hyper-parameter Sensitivity (RQ3)

We extensively evaluate the effect of hyper-parameters on the performance of NLCF, including sampling ratio α, graph sentence length l, and similarity threshold t. The results, presented in Figure 3, Figure 5, and Table 5, reveal two observations:

First, NLCF achieves optimal performance with a graph sentence length $l = 4$. This length indicates that third-order collaborative information suffices for high-quality recommendations, while higher-order information may introduce noise without contributing positively to performance. Increasing the sampling ratio improves model performance. Notably, performance declines only slightly when the ratio is lowered from 1 to 0.75, indicating the effectiveness of our sampling method.

Second, the similarity threshold demonstrates a critical role in sampling effectiveness. A moderate threshold value enables NLCF to maintain a more representative subset of samples. High threshold values impose overly strict similarity constraints, effectively reducing the sampling to random selection as most graph sentences are deemed dissimilar. Conversely, low threshold values lack discriminative power, marking most samples as similar and diminishing the sampling strategy's effectiveness.

Model	Steam		ML-10M		ML-1M	
	NDCG@5	NDCG@10	NDCG@5	NDCG@10	NDCG@5	NDCG@10
LightGCN (He et al. 2020)	0.2744	0.2790	0.1988	0.2007	0.2695	0.2436
LightGCL (Cai et al. 2023)	0.2623	0.2807	0.1944	0.2065	0.2112	0.2045
HMLET (Kong et al. 2022)	0.2730	0.2853	0.1919	0.1950	0.2723	0.2503
AFDGCF (Wu et al. 2024b)	0.2809	0.2897	0.2002	0.2004	0.2694	0.2484
TransRec (Lin et al. 2024b)	0.2796	0.2843	0.1961	0.1975	0.2706	0.2435
LLM-CF (Sun et al. 2024)	0.2772	0.2829	0.1976	0.1993	0.2735	0.2478
LETTER (Wang et al. 2024)	0.2717	0.2776	0.1928	0.1883	0.2683	0.2350
LC-Rec (Zheng et al. 2024)	0.2615	0.2682	0.1865	0.1822	0.2626	0.2321
LLMEmb (Liu et al. 2025)	0.2785	0.2858	0.2042	0.2034	0.2711	0.2446
NLCF	**0.2864**	**0.2948**	**0.2171**	**0.2147**	**0.2796**	**0.2558**

Table 1: NDCG performance with $N = 5$ and 10 across Steam, ML-10M, and ML-1M datasets.

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

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%

Table 3: Online A/B tests on the industrial platform.

Online Evaluation (RQ4)

We deploy NLCF on a billion-scale online shopping platform and conduct A/B testing to evaluate its performance. The platform serves hundreds of millions of users and billions of items, supported by two main components: an offline computing center and an online service center. The offline computing center processes user logs and generates a graph corpus from processed user interactions through distributed jobs. It trains NLCF and generates collaborative embeddings for each user and item. Notably, we do not require a tokenizer to handle billions of tokens for mapping. Another offline computing center task is the routine update of collaborative embeddings. This process involves generating a new graph corpus from recent user interactions on a daily basis. Because the final embeddings are constructed via summation, they can be updated incrementally by simply adding the hidden representations derived from this new corpus. The generated collaborative embeddings are then transmitted to the online service center, which uses these pre-computed user embeddings to efficiently retrieve items from a massive item pool, thereby eliminating the need for costly real-time LLM reasoning. NLCF is deployed as a recall model, replacing two baselines: the traditional graph CF model LightGCN and the LLM-based approach LLM-CF.

The performance metrics shown in Table 3 are averaged over eight consecutive weeks, with each model allocated 5% of online traffic. Compared to LightGCN, NLCF achieves significant improvements: +4.15% in PCTR, +3.11% in UCTR, and +5.78% in GMV, demonstrating its superiority in capturing user preferences and driving item consumption. Despite using higher-dimensional embeddings, NLCF incurs only a 1.46% increase in latency, ensuring scalability. Against LLM-CF, NLCF shows moderate gains: +2.51% in PCTR, +2.47% in UCTR, and +3.14% in GMV, confirming the benefits of learning collaborative embeddings over data augmentation. Additionally, NLCF reduces latency by 20.17% compared to LLM-CF, highlighting embedding-only inference efficiency.

Conclusion

This study introduces a novel paradigm for fine-tuning LLMs for CF task, enabling efficient modeling of multi-order interactions in the user-item graph to learn effective user and item embeddings. Existing methods that prompt LLMs with graph descriptions face two major limitations: (i) Efficiency – slower inference due to token-by-token item generation compared to embedding-based search, and (ii) Topological Modeling – difficulty encoding multi-order interactions and collaborative information from user-item graphs. To address these challenges, we propose NLCF, which treats the user-item graph as a new language. This approach is built on two insights: (1) LLMs excel at learning new languages with suitable corpora, and (2) token transition probabilities in language align with node transition probabilities in graphs. NLCF operates in two stages: (i) transforming the user-item graph into a language corpus that encodes multi-order interactions, and (ii) fine-tuning LLMs on this corpus to capture underlying multi-order collaborative information and construct collaborative user and item embeddings for efficient search. Extensive offline experiments demonstrate NLCF's superior performance over both LLM-based and traditional graph CF baselines while significantly improve computational efficiency over LLM-based baselines. And online A/B tests conducted on a world-leading shopping platform validate NLCF's effectiveness in real-world applications.

Acknowledgements

The work described in this paper was fully supported by a grant from the Innovation and Technology Commission of the Hong Kong Special Administrative Region, China (Project No. GHP/391/22).

References

Bao, K.; Zhang, J.; Zhang, Y.; Wang, W.; Feng, F.; and He, X. 2023. Tallrec: An effective and efficient tuning framework to align large language model with recommendation. In *Proceedings of the 17th ACM Conference on Recommender Systems*, 1007–1014.

Cai, X.; Huang, C.; Xia, L.; and Ren, X. 2023. LightGCL: Simple Yet Effective Graph Contrastive Learning for Recommendation. In *The Eleventh International Conference on Learning Representations*.

Chen, S.; Zhang, Q.; Dong, J.; Hua, W.; Li, Q.; and Huang, X. 2024. Entity Alignment with Noisy Annotations from Large Language Models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Chen, S.; Zhou, C.; Yuan, Z.; Zhang, Q.; Cui, Z.; Chen, H.; Xiao, Y.; Cao, J.; and Huang, X. 2025. You Don't Need Pre-built Graphs for RAG: Retrieval Augmented Generation with Adaptive Reasoning Structures. In *The Fortieth AAAI Conference on Artificial Intelligence*.

Grover, A.; and Leskovec, J. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 855–864.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30.

Harper, F. M.; and Konstan, J. A. 2016. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.*, 5(4): 19:1–19:19.

He, X.; Deng, K.; Wang, X.; Li, Y.; Zhang, Y.; and Wang, M. 2020. 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*, 639–648.

Hong, Z.; Yuan, Z.; Chen, H.; Zhang, Q.; Huang, F.; and Huang, X. 2024. Knowledge-to-SQL: Enhancing SQL Generation with Data Expert LLM. In *Findings of the Association for Computational Linguistics: ACL 2024*.

Hong, Z.; Yuan, Z.; Zhang, Q.; Chen, H.; Dong, J.; Huang, F.; and Huang, X. 2025. Next-generation database interfaces: A survey of llm-based text-to-sql. *IEEE Transactions on Knowledge and Data Engineering*.

Hu, E. J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; and Chen, W. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Huang, F.; Yang, Z.; Jiang, J.; Bei, Y.; Zhang, Y.; and Chen, H. 2024. Large Language Model Interaction Simulator for Cold-Start Item Recommendation. *arXiv preprint arXiv:2402.09176*.

Jiang, Y.; Yang, Y.; Xia, L.; Luo, D.; Lin, K.; and Huang, C. 2024. RecLM: Recommendation Instruction Tuning. *arXiv preprint arXiv:2412.19302*.

Kang, W.; and McAuley, J. J. 2018. Self-Attentive Sequential Recommendation. In *IEEE International Conference on Data Mining, ICDM 2018, Singapore, November 17-20, 2018*, 197–206. IEEE Computer Society.

Kim, S.; Kang, H.; Choi, S.; Kim, D.; Yang, M.; and Park, C. 2024. 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*, 1395–1406.

Kong, T.; Kim, T.; Jeon, J.; Choi, J.; Lee, Y.-C.; Park, N.; and Kim, S.-W. 2022. Linear, or non-linear, that is the question! In *Proceedings of the fifteenth ACM international conference on web search and data mining*, 517–525.

Liao, J.; Li, S.; Yang, Z.; Wu, J.; Yuan, Y.; Wang, X.; and He, X. 2024. Llara: Large language-recommendation assistant. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1785–1795.

Lin, J.; Shan, R.; Zhu, C.; Du, K.; Chen, B.; Quan, S.; Tang, R.; Yu, Y.; and Zhang, W. 2024a. Rella: Retrieval-enhanced large language models for lifelong sequential behavior comprehension in recommendation. In *Proceedings of the ACM on Web Conference 2024*, 3497–3508.

Lin, X.; Wang, W.; Li, Y.; Feng, F.; Ng, S.-K.; and Chua, T.-S. 2024b. 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*, 1816–1826.

Lin, X.; Wang, W.; Li, Y.; Yang, S.; Feng, F.; Wei, Y.; and Chua, T.-S. 2024c. Data-efficient Fine-tuning for LLM-based Recommendation. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 365–374.

Lin, Z.; Tian, C.; Hou, Y.; and Zhao, W. X. 2022. Improving graph collaborative filtering with neighborhood-enriched contrastive learning. In *Proceedings of the ACM web conference 2022*, 2320–2329.

Liu, Q.; Wu, X.; Wang, W.; Wang, Y.; Zhu, Y.; Zhao, X.; Tian, F.; and Zheng, Y. 2025. LLMEmb: Large Language Model Can Be a Good Embedding Generator for Sequential Recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, 12183–12191.

Qu, H.; Fan, W.; Zhao, Z.; and Li, Q. 2024. TokenRec: Learning to Tokenize ID for LLM-based Generative Recommendation. *arXiv preprint arXiv:2406.10450*.

Sun, Z.; Si, Z.; Zang, X.; Zheng, K.; Song, Y.; Zhang, X.; and Xu, J. 2024. Large language models enhanced collaborative filtering. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, 2178–2188.

Wang, W.; Bao, H.; Lin, X.; Zhang, J.; Li, Y.; Feng, F.; Ng, S.-K.; and Chua, T.-S. 2024. Learnable item tokenization for generative recommendation. In *Proceedings of the 33rd*

ACM International Conference on Information and Knowledge Management, 2400–2409.

Wang, X.; He, X.; Wang, M.; Feng, F.; and Chua, T.-S. 2019. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, 165–174.

Wei, W.; Ren, X.; Tang, J.; Wang, Q.; Su, L.; Cheng, S.; Wang, J.; Yin, D.; and Huang, C. 2024. Llmrec: Large language models with graph augmentation for recommendation. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, 806–815.

Wu, J.; Liu, Q.; Hu, H.; Fan, W.; Liu, S.; Li, Q.; Wu, X.-M.; and Tang, K. 2023. Leveraging Large Language Models (LLMs) to Empower Training-Free Dataset Condensation for Content-Based Recommendation. *arXiv preprint arXiv:2310.09874*.

Wu, L.; Qiu, Z.; Zheng, Z.; Zhu, H.; and Chen, E. 2024a. Exploring large language model for graph data understanding in online job recommendations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 9178–9186.

Wu, W.; Wang, C.; Shen, D.; Qin, C.; Chen, L.; and Xiong, H. 2024b. 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*, 1242–1252.

Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; and Philip, S. Y. 2020. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1): 4–24.

Yang, H.; Wang, X.; Tao, Q.; Hu, S.; Lin, Z.; and Zhang, M. 2024. GL-Fusion: Rethinking the Combination of Graph Neural Network and Large Language model. *arXiv preprint arXiv:2412.06849*.

Yang, Z.; Wu, J.; Luo, Y.; Zhang, J.; Yuan, Y.; Zhang, A.; Wang, X.; and He, X. 2023. Large language model can interpret latent space of sequential recommender. *arXiv preprint arXiv:2310.20487*.

Yuan, Z.; Chen, H.; Hong, Z.; Zhang, Q.; Huang, F.; Li, Q.; and Huang, X. 2025. Knapsack optimization-based schema linking for llm-based Text-to-SQL generation. *arXiv preprint arXiv:2502.12911*.

Yuan, Z.; Yuan, F.; Song, Y.; Li, Y.; Fu, J.; Yang, F.; Pan, Y.; and Ni, Y. 2023. Where to go next for recommender systems? id-vs. modality-based recommender models revisited. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2639–2649.

Zhang, A.; Deng, Y.; Lin, Y.; Chen, X.; Wen, J.-R.; and Chua, T.-S. 2024a. Large Language Model Powered Agents for Information Retrieval. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2989–2992.

Zhang, Q.; Chen, S.; Bei, Y.; Yuan, Z.; Zhou, H.; Hong, Z.; Chen, H.; Xiao, Y.; Zhou, C.; Dong, J.; et al. 2025. A survey of graph retrieval-augmented generation for customized large language models. *arXiv preprint arXiv:2501.13958*.

Zhang, Y.; Bao, K.; Yan, M.; Wang, W.; Feng, F.; and He, X. 2024b. Text-like Encoding of Collaborative Information in Large Language Models for Recommendation. *arXiv preprint arXiv:2406.03210*.

Zhang, Y.; Feng, F.; Zhang, J.; Bao, K.; Wang, Q.; and He, X. 2023. Collm: Integrating collaborative embeddings into large language models for recommendation. *arXiv preprint arXiv:2310.19488*.

Zhao, W. X.; Zhou, K.; Li, J.; Tang, T.; Wang, X.; Hou, Y.; Min, Y.; Zhang, B.; Zhang, J.; Dong, Z.; et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*.

Zheng, B.; Hou, Y.; Lu, H.; Chen, Y.; Zhao, W. X.; Chen, M.; and Wen, J.-R. 2024. 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)*, 1435–1448. IEEE.

Zhou, C.; Du, J.; Zhou, H.; Chen, H.; Huang, F.; and Huang, X. 2025a. Text-Attributed Graph Learning with Coupled Augmentations. In *Proceedings of the 31st International Conference on Computational Linguistics*, 10865–10876.

Zhou, C.; Wang, Z.; Chen, S.; Du, J.; Zheng, Q.; Xu, Z.; and Huang, X. 2025b. Taming language models for text-attributed graph learning with decoupled aggregation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 3463–3474.

Zhou, H.; Chen, H.; Dong, J.; Zha, D.; Zhou, C.; and Huang, X. 2023. Adaptive popularity debiasing aggregator for graph collaborative filtering. In *Proceedings of the 46th international ACM SIGIR conference on research and development in information retrieval*, 7–17.

Zhou, H.; Du, J.; Zhou, C.; Yang, C.; Xiao, Y.; Xie, Y.; and Huang, X. 2025c. Each Graph is a New Language: Graph Learning with LLMs. *arXiv preprint arXiv:2501.11478*.

Zhou, H.; Yu, K.; Zhang, Q.; Chen, H.; Zha, D.; Pei, W.; Kong, A.; and Huang, X. 2025d. Self-Monitoring Large Language Models for Click-Through Rate Prediction. *ACM Transactions on Information Systems*, 44(1): 1–25.

Zhou, H.; Zhou, S.; Chen, H.; Liu, N.; Yang, F.; and Huang, X. 2024. Enhancing explainable rating prediction through annotated macro concepts. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 11736–11748.

Zhuang, L.; Chen, S.; Xiao, Y.; Zhou, H.; Zhang, Y.; Chen, H.; Zhang, Q.; and Huang, X. 2025. LinearRAG: Linear Graph Retrieval Augmented Generation on Large-scale Corpora. *arXiv preprint arXiv:2510.10114*.

A. Proof for Theorem 1

We show that the empirical conditional probability $P_{\mathrm{s}}(v_i \mid v_u)$, computed from the corpus, converges to the transition probability $P_{\mathrm{g}}(v_i \mid v_u)$ in graphs.

Let $N_{\mathrm{s}}(v_u \to v_i)$ denote the number of transitions from v_u to v_i observed in the initial graph corpus. The empirical conditional probability is defined as:

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

where $\hat{\mathcal{N}}(v_u)$ represents the multi-order neighboring nodes within the walk length l. And the expected number of transitions from v_u to v_j, denoted $\mathbb{E}[N_{\mathrm{s}}(v_u \to v_j)]$, 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 and $k \to \infty$, the observed transition count $N_{\mathrm{s}}(v_u \to v_i)$ converges to its expectation:

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

The total number of transitions from v_u in the sampled walks is: $\sum_{j \in \hat{\mathcal{N}}(v_u)} N_{\mathrm{s}}(v_u \to v_j)$. The expectation of this term is:

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

Substituting these results into the definition of $P_{\mathrm{s}}(v_i \mid v_u)$, we have:

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

And by the Law of Large Numbers, the numerator and denominator converge to their expectations:

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

This proves that the random walk sampling process faithfully align the graph's transition probabilities with the empirical conditional probability in graph corpus.

B. Proof for Theorem 2

This section 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:

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

B.1 Single Hash Function Analysis

For $k = 1$, we establish the following lemma:

Lemma 4. *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:*

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

Proof. 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 $\mathrm{sig}(s_p) = \min_{v_i \in s_p} h(v_i)$ represent the first element of s_p in this permutation. The equality $\mathrm{sig}(s_p) = \mathrm{sig}(s_q)$ 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: $P(\mathrm{sig}(s_p) = \mathrm{sig}(s_q)) = \frac{|s_p \cap s_q|}{|s_p \cup s_q|} = J(s_p, s_q)$. $\square$

B.2 Multiple Hash Functions Analysis

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:

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

Theorem 5. *Given k independent hash functions, the MinHash estimator $\hat{J}(s_p, s_q)$ is unbiased with variance $\frac{J(s_p,s_q)(1-J(s_p,s_q))}{k}$.*

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

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

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

$\square$

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.

C. Related Work

We broadly categorize LLMs in recommender system into two groups: LLMs as Recommender and LLMs as Data Augmentor.

C.1 LLMs as Recommender

Recent research has increasingly explored LLMs' potential as recommenders (Zhang et al. 2024a). Given LLMs' natural language processing capabilities (Chen et al. 2025; Hong et al. 2025; Yuan et al. 2025), there are two lines of approaches.

Table 4: Precision performance with $N = 5$ and 10 across Steam, ML-10M, and ML-1M datasets.

Model	Steam		ML-10M		ML-1M	
	Precision@5	Precision@10	Precision@5	Precision@10	Precision@5	Precision@10
LightGCN (He et al. 2020)	0.1232	0.0813	0.1559	0.1195	0.2527	0.2153
LightGCL (Cai et al. 2023)	0.0654	0.0503	0.1248	0.0911	0.1873	0.1569
HMLET (Kong et al. 2022)	0.1258	0.0884	0.1461	0.1141	0.2589	0.2238
AFDGCF (Wu et al. 2024b)	0.1307	0.0909	0.1575	0.1208	0.2575	0.2223
TransRec (Lin et al. 2024b)	0.0858	0.0448	0.1081	0.0603	0.1766	0.1218
LLM-CF (Sun et al. 2024)	0.1257	0.0827	0.1524	0.1132	0.2550	0.2170
LETTER (Wang et al. 2024)	0.1033	0.0690	0.1213	0.0835	0.2018	0.1645
LC-Rec (Zheng et al. 2024)	0.0924	0.0606	0.1106	0.0664	0.1931	0.1536
LLMEmb (Liu et al. 2025)	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**

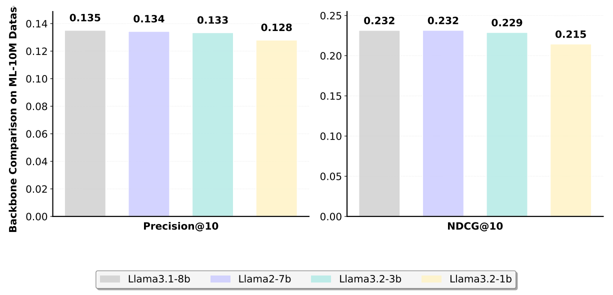


Figure 4: The effect of replacing LLMs with different parameter scales on the performance of the NLCF model on the ML-10M dataset.

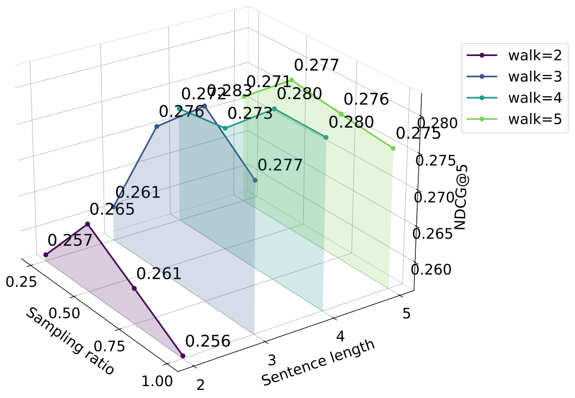


Figure 5: The impact of graph sentence length l and sampling ratio α on NDCG@5 on ML-1M dataset.

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

Table 5: The effect of similarity threshold t on the performance of NLCF on the ML-1M dataset.

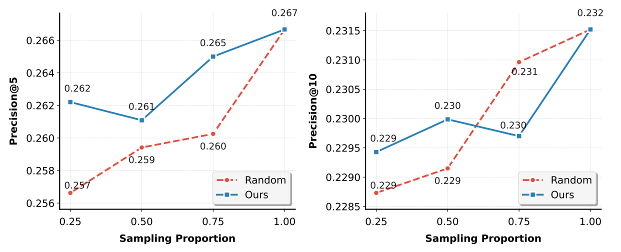


Figure 6: The impact of different sampling strategies on NLCF performance: including random sampling, and our similarity-based sampling across ML-1M dataset.

Description Reasoning-based Methods. One direct approach is to leverage the LLM's inherent natural language understanding and reasoning capabilities. These methods transform user-item interactions into descriptive natural language prompts and then task the LLM with prediction. For a given user, his interaction history is formatted into a sentence. For instance, methods like TALLRec (Bao et al. 2023) organize item IDs or titles into simple descriptive sequences. The core idea is to rely on the LLM's reasoning power to infer user preferences from this textual data. More advanced studies in this category attempt to incorporate richer context. For example, some pioneering work retrieves user and item text along selected graph paths to provide the LLM with more graph-specific information to reason over (Wu et al. 2024a). A prompt describing a single user's history lacks the powerful signal that arises from the aggregated behavior of thousands of similar users. For instance, the fact that many users who interact with item A also interact with item B is a strong collaborative pattern that is lost in a simple textual description. Furthermore, these methods depend on slow, token-by-token text generation for inference, making them inefficient for real-time recommendations over large item sets.

Embedding and Structural Information Injection Methods. To address the shortcomings of pure text-based reasoning, a second line of work aims to inject richer collaborative information directly into the LLM (Zhou et al. 2025a,b). This is achieved by incorporating pre-computed embeddings or creating additional tokens for structural elements. These methods typically involve a two-stage process. First, a traditional model, often a Graph CF model, is used to learn latent user and item embeddings from the full user-item graph. These embeddings, which encode collaborative signals, are then injected into the LLM's input. For example, studies like RecInterpreter (Yang et al. 2023), LLARA (Liao et al. 2024) and CoLLM (Zhang et al. 2023) plug these pre-computed embeddings directly into the prompt. To better integrate these numerical representations, BinLLM (Zhang et al. 2024b) proposed converting the continuous values in embeddings into discrete numerical tokens that are more natural for LLMs to process. Recognizing the LLM's ability to interpret latent representations through fine-tuning, researchers begin incorporating these representations more deeply, e.g., developing tailored tokenizers (Kim et al. 2024). Instead of injecting embeddings from an external model, these methods learn to tokenize these embeddings into several tokens and incorporate them into the LLM's vocabulary, e.g., LETTER (Wang et al. 2024) and TokenRec (Qu et al. 2024). NLCF offers two distinct advantages over these approaches: (i) it generates collaborative embeddings for efficient item recall across the entire item set, rather than relying on computationally intensive text generation for a limited item subset, and (ii) it directly learns collaborative information from graphs, fully leveraging LLMs' capabilities.

C.2 LLMs as Data Augmentor

Directly using LLMs as a recommender may introduce severe latency. Consequently, many approaches utilize LLMs as data augmenters to enhance representation learning of traditional recommendation models, addressing the semantic limitations of ID embeddings. A fundamental approach uses LLMs to integrate user and item attributes into unified vectors (Wei et al. 2024). However, this method under-utilizes LLMs' reasoning capabilities regarding user behavior patterns. More advanced approaches employ LLMs as interaction simulators, leveraging their step-by-step natural language reasoning abilities to achieve promising results, even in cold-start scenarios (Huang et al. 2024). LLMs can also generate augmented user sequences for traditional recommendation models (Sun et al. 2024) and produce valuable side information, such as natural language explanations for interactions (Zhou et al. 2024), enabling more informative recommendations. RecLM addresses the challenge of noisy initial profiles by generating refined, high-quality user and item profiles (Jiang et al. 2024). Our method distinguishes itself by directly leveraging LLMs' capability to estimate user preferences while avoiding excessive computational overhead by learning collaborative embeddings.

D. Preliminary Details

Low Rank Adaption. Low-Rank Adaptation (LoRA) is a widely used parameter-efficient fine-tuning method for LLMs. LoRA incorporates trainable matrices into LLM (Hu et al. 2021) through low-rank decomposition. Specifically, given the weight matrix of a LLM, denoted as $\mathbf{W} \in \mathbb{R}^{d\times d'}$, LoRA introduces a modification $\hat{\mathbf{W}} = \mathbf{W} + BA$, which represents the low-rank adaptation for fine-tuning.

Collaborative Filtering. CF task learns user and item embeddings to predict user preferences based on the user-item interactions. And multi-order interactions and underlying collaborative information from user-item graph $\mathcal{G}$ (He et al. 2020) are beneficial for mining user preferences. During training, CF models update user embeddings $\mathbf{h}_{v_u}$ and item embeddings $\mathbf{h}_{v_i}$ to model multi-order interactions. The initial embeddings $\mathbf{H}$ is updated as:

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

where $f(\cdot)$ is a Graph Neural Network (GNN) model that captures high-order collaborative information underlying user-item interactions. The model optimizes these embeddings to effectively predict observed interactions in $\mathcal{E}$.

During inference, the model predicts user preferences by leveraging learned embeddings to search the entire item set $\mathcal{I}$. For each user v_u, preference scores $\hat{y}_{ui}$ are computed for all items $v_i \in \mathcal{I}$ by using a simple dot product:

$$\hat{y}_{ui} = \mathbf{h}_{v_u}^{(k)\top}\mathbf{h}_{v_i}^{(k)}. \tag{10}$$

The model then ranks items by preference scores and recommends the top-N items to each user.

E. Dynamic Memory Bank Details

Fine-tuning LLMs on large-scale datasets introduces significant memory challenges, as each user and item token requires storing its embedding and associated weight vectors in the head layer $\mathbf{W}_p \in \mathbb{R}^{|V|\times d}$. To address these scalability issues, we design a dynamic memory bank mechanism that efficiently manages these vectors during fine-tuning. Our dynamic memory bank mechanism consists of two key strategies: (i) Batch-wise embedding loading: Instead of loading all user and item embeddings and head-layer weight vectors into memory, NLCF loads only the embeddings and weights associated with tokens in the current batch. This significantly reduces memory overhead. (ii) Sampled softmax optimization: To further reduce memory usage, we optimize only the weight vectors corresponding to tokens in the current batch, instead of the full set of head-layer weights. By integrating these strategies, the dynamic memory bank mechanism enables NLCF to efficiently fine-tune LLMs on large-scale datasets while maintaining scalability.

F. Experimental Details

F.1 Dataset Details

We evaluate NLCF on three large-scale real-world datasets:

- **ML-10M**: A dataset from MovieLens, i.e., GroupLens Research, including rich movie metadata such as titles and genres.
- **Steam**: A gaming platform dataset containing user-game interactions and game metadata, including titles, publishers, etc.
- **ML-1M**: A dataset from MovieLens containing movie ratings, user and item metadata.

These three datasets are chosen since they contain user/item textual attributes, facilitating LLM baseline running. Following previous efforts (Lin et al. 2022), we split each dataset into training, validation, and test sets with a ratio of 8:1:1. We filter out users and items from the test and validation set that do not appear in the training set. Following prior LLM-based recommendation studies (Lin et al. 2022), we only retain interactions whose ratings are greater than or equal to 3. Following (Wang et al. 2019), we remove users and items with fewer than 10 interactions.

F.2 Baseline Details

The introduction of baselines is listed below:

- **LightGCN** (He et al. 2020): A lightweight framework that learns user and item embeddings through linear graph propagation.
- **LightGCL** (Cai et al. 2023): A graph-based model that leverages singular value decomposition for data augmentation to address sparsity and bias issues.
- **HMLET** (Kong et al. 2022): A hybrid graph model combining linear and non-linear embedding propagation, with adaptive propagation based on node centrality.
- **AFDGCF** (Wu et al. 2024b): A dynamic feature-dimension filtering approach that addresses over-correlation and over-smoothing in graph CF.
- **TransRec** (Lin et al. 2024b): A transition-based framework utilizing multi-facet identifiers for enhanced item indexing and ranking.
- **LLM-CF** (Sun et al. 2024): An instruction-tuning approach that leverages LLMs' world knowledge for graph CF.
- **LETTER** (Wang et al. 2024): A learnable tokenization framework incorporating semantics from LLMs and collaborative information from graph CF models.
- **LC-Rec** (Zheng et al. 2024): A learnable tokenization framework that fine-tunes LLMs on various prompt templates.
- **LLMEmb** (Liu et al. 2025): A framework fine-tunes LLMs on user and item attributes for generating better user and item embeddings for downstream models.

The baselines are divided into two groups: traditional graph CF baselines and LLM-based baselines. Among the LLM-based baselines, we include two from the description reasoning-based category, i.e., LLM-CF and TransRec, and two from the embedding injection-based category, i.e., LETTER and LC-Rec. Additionally, we include two baselines that fine-tune LLMs for side information generation to achieve efficient inference , i.e., LLM-CF and LLMEmb.

F.3 Implementation Details

The underlying GPU computations is managed by CUDA 12.8. For a fair comparison, we measure running time on a single GPU, distinguishing between training and test phases. For methods that pre-compute user and item embeddings, we search target items by performing similarity search based on these embeddings. For generation-based methods, we generate target item identifiers using trained LLMs. For NLCF, we set the batch size to 256 with a learning rate of 1e-4. Traditional graph CF baselines use a batch size of 4096. While the A100 supports larger batches, we find this value optimal for balancing gradient stability and convergence speed. For LLM-based baselines, we maximize their batch size within hardware constraints, as prompt lengths vary across models. We tune hyper-parameter using grid searches based on the performance of validation set. The graph sentence length l is selected from $\{2, 3, 4, 5\}$, similarity threshold t from $\{0.25, 0.5, 0.75, 1\}$, and aggregation position k is set as 1 to avoid redundancy. The sampling proportion α is fixed as 75% except for the ablation study experiments. For the main comparison, we adopt LLAMA-3.1-1B as the default LLM backbone to ensure a fair comparison. To evaluate the generalization capability of NLCF, we further evaluate it with several LLAMA variants, including LLAMA-2-7B, LLAMA-3.2-3B, and LLAMA-3.1-8B. To control for the randomness, each model gets run five times. And the reported results have passed the significance test with p-value < 0.01. For offline experiments, we use the full-ranking evaluation approach Precision@N and NDCG@N following previous approaches, where N is set to 5 and 10. For industrial A/B tests, we use the online metrics: PCTR, UCTR, GMV and ResTime.

F.4 Evaluation Metric Details

The introduction of evaluation metrics is listed below:

- **Precision** measures the proportion of recommended items that are actually relevant to the user in the top K recommendations.
- **NDCG** evaluates the ranking quality of recommendations by considering both the relevance and the order of the recommended items.
- **PCTR** is the ratio of overall clicks to overall impressions.
- **UCTR** is the ratio of users who click to overall.
- **GMV** is the total values of user purchases.
- **ResTime** measures the average response time from user request phase and return recommendation results phase.

F.5 NLCF Pseudocode

The NLCF model, whose pseudocode is detailed in Algorithm 1, enhances collaborative filtering by leveraging LLMs through three main phases:

In the *graph corpus collection phase*, NLCF represents user-item interactions as a bipartite graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \mathcal{U} \cup \mathcal{I}$. For each node $v_u \in \mathcal{V}$, the algorithm performs d_u random walks of length l, generating graph sentences. These sentences are clustered using MinHash signatures based on similarity threshold t. A sampling probability $P(s_q) \propto 1/|\mathcal{C}_i|$ prioritizes sentences from smaller clusters, producing an optimized corpus $\mathcal{S}'$ of size $\alpha \cdot |\mathcal{S}|$.

During the *collaborative embedding construction phase*, NLCF initializes LoRA parameters $\hat{\mathbf{W}}$ and fine-tunes the LLM by minimizing the prediction loss $L_{pre} = -\sum_q \sum_j \log P(s_{q,j}|s_{q,<j}, \mathbf{W}p, \hat{\mathbf{W}})$. Using a dynamic memory bank, it loads only necessary embeddings for each

Algorithm 1: New Language Collaborative Filtering (NLCF)

```
1: Input: User-item graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \mathcal{U} \cup \mathcal{I}$, LLM with parameter $\mathbf{W}_p$, Sampling ratio $\alpha$, walk length $l$, similarity threshold $t$, Aggregation position $k$
2: Output: User embeddings $\mathbf{h}_{v_u}$, Item embeddings $\mathbf{h}_{v_i}$
3: procedure CollectGraphCorpus($\mathcal{G}, l, \alpha, t$)
4:     $\mathcal{S} \leftarrow \emptyset$                              ▷ Initial corpus
5:     for each node $v_u \in \mathcal{V}$ do
6:         $d_u \leftarrow$ degree of $v_u$
7:         for $i = 1$ to $d_u$ do
8:             $s \leftarrow$ RandomWalk($\mathcal{G}, v_u, l$)      ▷ Walk of length $l$ starting from $v_u$
9:             $\mathcal{S} \leftarrow \mathcal{S} \cup \{s\}$
10:        end for
11:    end for
12:    Compute MinHash signatures for all $s \in \mathcal{S}$
13:    Insert graph sentences into hash table $\mathcal{H}$ and group into clusters $\mathcal{C}_i$ based on similarity threshold $t$
14:    for each cluster $\mathcal{C}_i$ do
15:        for each sentence $s_q \in \mathcal{C}_i$ do
16:            $P(s_q) \leftarrow \frac{1/|\mathcal{C}_i|}{\sum_j 1/|\mathcal{C}_j|}$      ▷ Calculate sampling probabilities
17:        end for
18:    end for
19:    $\mathcal{S}' \leftarrow$ Sample $\alpha \cdot |\mathcal{S}|$ sentences from $\mathcal{S}$ according to $P(s_q)$
20:    return $\mathcal{S}'$
21: end procedure
22: procedure ConstructEmbeddings($\mathcal{S}', \mathbf{W}_p$)
23:    Initialize LoRA parameters $\hat{\mathbf{W}}$
24:    while not converged do
25:        Sample batch of graph sentences $\{s_q\}$ from $\mathcal{S}'$ of size $b$
26:        Load embeddings and weights for tokens in batch using dynamic memory bank
27:        $L_{pre} = -\sum_q \sum_j \log P(s_{q,j} | s_{q,<j}, \mathbf{W}_p, \hat{\mathbf{W}})$   ▷ Compute loss
28:        Update $\hat{\mathbf{W}}$ to minimize $L_{pre}$
29:    end while
30:    for each user $v_u \in \mathcal{U}$ do
31:        $\mathcal{K} \leftarrow \{p : s_q \in \mathcal{S}', s_{q,p} = v_u, p = |s_q| - k, 0 < k < |s_q|\}$
32:        $\mathbf{h}_{v_u} \leftarrow \sum_{p \in \mathcal{K}} \text{LLM}(\{s_{q,1}, s_{q,2}, ..., s_{q,p-1}\})$
33:    end for
34:    for each item $v_i \in \mathcal{I}$ do
35:        $\mathcal{K} \leftarrow \{p : s_q \in \mathcal{S}', s_{q,p} = v_i, p = |s_q| - k, 0 < k < |s_q|\}$
36:        $\mathbf{h}_{v_i} \leftarrow \sum_{p \in \mathcal{K}} \text{LLM}(\{s_{q,1}, s_{q,2}, ..., s_{q,p-1}\})$
37:    end for
38:    return $\mathbf{h}_{v_u}, \mathbf{h}_{v_i}$
39: end procedure
40: procedure NLCF($\mathcal{G}, \mathbf{W}_p, l, \alpha, t, k, b$)
41:    $\mathcal{S}' \leftarrow$ CollectGraphCorpus($\mathcal{G}, l, \alpha, t$)
42:    $\mathbf{h}_{v_u}, \mathbf{h}_{v_i} \leftarrow$ ConstructEmbeddings($\mathcal{S}', \mathbf{W}_p$)
43:    return $\mathbf{h}_{v_u}, \mathbf{h}_{v_i}$
44: end procedure
45: procedure Recommend($v_u, \mathcal{I}, \mathbf{h}_{v_u}, \mathbf{h}_{v_i}, N$)
46:    for each item $v_i \in \mathcal{I}$ do
47:        $\hat{y}_{ui} \leftarrow \mathbf{h}_{v_u}^\top \cdot \mathbf{h}_{v_i}$                 ▷ Dot product
48:    end for
49:    return Top-$N$ items with highest $\hat{y}_{ui}$
50: end procedure
```

batch. For users $v_u \in \mathcal{U}$ and items $v_i \in \mathcal{I}$, the model aggregates representations from positions where they appear in graph sentences, generating embeddings $\mathbf{h}v_u$ and $\mathbf{h}_{v_i}$.

In the *recommendation phase*, NLCF computes the similarity between user v_u and each item v_i as $\hat{y}ui = \mathbf{h}{v_u}^\top \cdot \mathbf{h}_{v_i}$, returning the top-$N$ items with highest scores.

F.6 Limitations

One limitation of NLCF is the need to tune multiple hyperparameters, including sentence length, position step, and similarity threshold. Identifying the optimal configuration requires multiple grid or random searches, which can be laborious despite the training and inference efficiency of our proposed model. Another limitation stems from the inherent requirement of LLMs, which demand higher GPU resources compared to traditional CF models, even though our model is designed for efficient training and inference.

F.7 Future Directions

Our future work will first focus on enhancing the interpretability aspects of NLCF by developing methods to extract human-understandable patterns from the learned LLM representations, potentially enabling explanation generation for recommendations. Second, investigating mechanisms to incorporate explicit user feedback for continual learning would enable the model to adapt to evolving user preferences while maintaining interpretability.